(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
29 September 2016 (29.09.2016)

WIPO | PCT

(10) International Publication Number
WO 2016/154001 A1

(51) International Patent Classification:
G06F 11/30 (2006.01) *G06F 12/14* (2006.01)

(21) International Application Number:
PCT/US2016/023142

(22) International Filing Date:
18 March 2016 (18.03.2016)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

62/136,340	20 March 2015 (20.03.2015)	US
62/136,385	20 March 2015 (20.03.2015)	US

(71) Applicant: RIVETZ CORP. [US/US]; 1209 Orange Street, Wilmington, DE 19801 (US).

(72) Inventors: SPRAGUE, Michael; 73 Bedford Street, New York, NY 10014 (US). SPRAGUE, Steven; 111 Swamp Road, Richmond, MA 02154 (US).

(74) Agents: FESSENDEN, Giovanna, H. et al.; Hamilton, Brook, Smith & Reynolds, P.C., 530 Virginia Rd, P.O. Box 9133, Concord, MA 01742-9133 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report (Art. 21(3))*

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments (Rule 48.2(h))*

(54) Title: AUTOMATED ATTESTATION OF DEVICE INTEGRITY USING THE BLOCK CHAIN

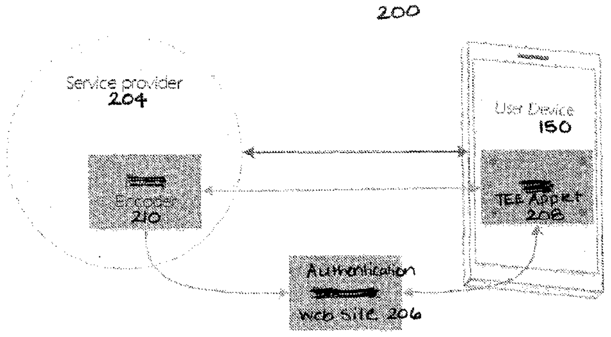


FIG. 2A: example device authentication system according to the invention.

(57) Abstract: Systems and methods are disclosed that provide for a full validation of an unknown client device prior to acceptance of a block chain transaction would provide further security for block chain transactions. The health of the device can be attested to prior to engaging in electronic transactions. In some embodiments, automation of full device integrity verification is provided as part of a block chain transaction. Certain aspects of the invention enable trust in devices. Some embodiments operate on the fundamental premise that a reliable relationship with a device can make for a much safer, easier and stronger relationship with an end user. Achieving this requires knowing with confidence that a device involved in a current transaction is the same device it was in previous transactions.

WO 2016/154001 A1

AUTOMATED ATTESTATION OF DEVICE INTEGRITY USING THE BLOCK CHAIN

RELATED APPLICATION

[0001] This application claims the benefit of U.S. Provisional Application No., 62/136,340 filed on March 20, 2015 and U.S. Provisional Application No., 62/136,385 filed on March 20, 2015. The entire teachings of the above applications are incorporated herein by reference.

BACKGROUND

[0002] The advent of decentralized transaction systems such as Bitcoin has provided the Internet with a reliably secure protocol for recording ownership over digital value known as the block chain. The system is rooted in private keys that enable people to exercise that digital value. However, when these keys are stored digitally, and particularly when they are transacted, they are vulnerable to theft which can result in substantial losses. Industry has for years anticipated a need for high-assurance operations in endpoint devices. Already deployed hardware security can be used to enhance the security and privacy for interactions between people and the block chain.

[0003] The block chain behind Bitcoin, the common ledger that is built on the backs of thousands of peered servers, is devised to be mathematically impenetrable. As long as a majority of participating peers act in support of the community one cannot leverage enough compute power to edit records of the past and thus steal value. With such a large community maintaining its integrity, it is deemed that only a vulnerability in elliptic curve cryptography could compromise the block chain. However, while the block chain itself is well secured, how an individual transacts with it is either very complex or subject to a number of well-known malware attacks. The result is that the quality of the instructions to the block chain are critical to assuring the quality of the protected transaction ledger.

SUMMARY

[0004] Most of the transactions captured in the Bitcoin block chain record a transfer of value from one person to another. Public keys represent the parties involved. Corresponding private keys enable a participant to claim the result. As there is no other method of oversight or control, it is paramount that the private key be secured. The block chain is an ephemeral construct. People can only interact with it through their control of a network connected device. Broadly speaking there are three ways in which this takes place. A) The person controls a machine that is itself a peer and writes directly into the block chain. B) The person uses a web site or mobile app to instruct a server acting on their behalf, or C) the person uses a web site or app to propagate a transaction that is locally formed.

[0005] In general, a private key is applied to sign a request. The execution environment is responsible for the accuracy of the request and protection of the private key. Attestation to the health and origin of the execution environment establishes its reliability.

[0006] There are a number of widespread tools that can be leveraged for improving the security of the execution environment. This ranges from hardware backed device identity to full trusted execution environments. The consumer web is the most broadly distributed services platform that is constructed on user identification methods rather than device identification. Unlike mobile telephony or cable television, for example, where service is authenticated by the enabling device, the web requires that end-users conduct the identification protocol, i.e. enter username and password. While there are benefits to the portability of this method, it is dangerously susceptible in practice. Users are terrible at remembering complex passwords and irritated by repetitive requests. The result is passwords like “GoYanks” and session keys that are allowed to persist for days. A device, on the other hand, will happily engage in a cryptographic authentication well beyond the capacity of any human with any of thousands of credentials stored in its hardware. And it will do it over and over without fatigue.

[0007] Except in extreme circumstances, portability in the form of username/password, has a role to play. But most of the time users engage with the same devices for the same interactions. By leveraging the devices they own to conduct basic authentication this consistency can be rewarded with immediate access for users and increased assurance for service providers.

[0008] The Internet is largely accessed by multi-purpose devices. PC’s, Tablets and Phones may host hundreds of applications and the vibrant market for new apps drives a very open environment. This is very user friendly until one of those apps disguises a malicious intent and begins to vandalize or steal from the other apps on the device. In addition to knowing whether the device is the same one it was before, a service provider should ask it, are you in the same state as before. When significant changes are known to have occurred this can indicate a potential threat. This knowledge enables service providers to take remedial action or at least request further confirmation from the device operator that the machine is still safe.

[0009] The user will often not know if their device is compromised, but if it can be detected, for example, that the BIOS has changed, a service can take cautionary steps.

[0010] Installing and running apps is meant to be very simple. However, there is a class of apps that can benefit greatly from strong assurance of their origin and opaque separation from the execution of other apps. This may be, for example, a Trusted Execution Environment or TEE. Unlike an app running on the primary OS and memory stack, an app running in a TEE can have access to cryptographic primitives that can be exercised without snooping by the OS. In ideal circumstances it also has direct access to user input and display to ensure a private interaction with the operator of the device.

[0011] Both proprietary and standards based solutions in support of device security have worked their way into the supply chain. The Trusted Platform Module, or TPM, for instance, is a security chip embedded on the motherboard of most modern PC’s. The technology is specified by the Trusted Computing Group (TCG), a non-profit consortium of dozens of major vendors. It was designed largely in support of enterprise network security but has a huge role to play in simplifying the consumer web. TPM’s having be shipping for half a dozen years and are now widely prevalent in modern PC’s. Microsoft logo compliance beginning in 2015 will further ensure that no machine is delivered without a TPM.

[0012] A TPM is relatively simple. It serves three basic purposes: PKI, BIOS integrity and encryption. While the technology has been pursued for well over a decade, it is only recently that devices with support for a TEE have become available. Intel began delivery of commercial solutions in 2011 and Trustonic launched in 2013. The platforms and associated tools are reaching the level of maturity required for consumer use. Deploying an app into a TEE is akin to delivering a dedicated hardware device. Execution and data are cryptographically isolated from any other function of the host.

[0013] The chip has no identity of its own, but can be asked to generate key pairs. AIK's, or Attestation Identity Keys, can be marked as "non-migratable" so that the private half of the key pair will never be visible outside the hardware. This provides an opportunity to establish a machine identity that cannot be cloned. Currently deployed TPM's, version 1.2, are limited to RSA and SHA-1. Version 2.0, coming soon, will be much more agile. The TPM also implements an Endorsement Key (EK). The EK is installed during manufacture and can be used to prove that the TPM is in a fact a real TPM. A system supporting a TPM will load Platform Configuration Registers (PCR's) during its boot sequence. Beginning with the firmware, each step in the boot process measures its state and the state of the next process and records a PCR value. As the PCR's are captured in the tamperproof TPM a reliable "quote" of the system's BIOS integrity can subsequently be requested. A PCR doesn't capture what actually happened it only captures, through a series of hashes, that nothing is changed. This is particularly important for protection against the most serious and otherwise undetectable attacks where a hacker compromises the machine bios or installs a secret hypervisor. Combined with an assurance signature from virus scanning software, one can establish a reliable state of machine health. TPM's also provide bulk encryption services. Encryption keys are generated in the TPM, but not stored there. Instead they are encrypted with a TPM bound Storage Root Key and returned to the requesting process. A process wishing to encrypt or decrypt a blob of data will first mount the desired key. The key is then decrypted in the hardware and made available for ciphering. As with most TPM keys, encryption keys can be further protected with a password if desired.

[0014] Trustonic (http://www.trustonic.com) is a joint venture of ARM, G+D and Gemalto. Trustonic provides a trusted execution environment across a broad array of smart devices. The goal is to enable the secure execution of sensitive application services. Trustonic is an implementation of the Global Platform standard for Trusted Execution Environments. Apps written to execute in the Trustonic TEE are signed and measured. Devices supporting Trustonic provide an isolated execution kernel so that a loaded app cannot be spied on by any other process running on the device, including debug operations on a rooted device. Trustonic was formed in 2012 and now ships with half a dozen manufactures and supports a couple dozen service providers. Over 200 million devices have now shipped with Trustonic support.

[0015] Intel vPro is collection of technologies built into modern Intel chip set. New machines marketed with vPro support the Intel TXT Trusted Execution Technology. Intel offers a secure processing environment in the Management Engine (ME) that enables protected execution of numerous cryptographic functions. One use of this capability has been the deployment of TPM 2.0 functionality implemented as an app in the ME. The Management Engine also supports secure display functions for conducting fully isolated communications with the user. In this manner an app executing in the ME can take direction from the user with a substantially reduced risk of compromise.

[0016] ARM TrustZone provides the silicon foundations that are available on all ARM processors. The primitives isolate a secured world of execution from the common execution space. ARM provides the designs that are then built into a number of standard processors. To take advantage of TrustZone, apps can either be deployed as part of system firmware by the manufacturer or can be delivered after the fact through third party tools like Trustonic, Linaro or Nvidia's open source micro kernel.

[0017] Some embodiments of the present invention apply these technologies into a set of services for enhancing the transaction environment that connects people and the block chain.

[0018] The concept of second factor authentication is well established though in limited use. It is perhaps utilized most prominently by Bitcoin service sites, where breaching a login can provide immediate and irreversible theft of funds. Most people are familiar with second factor in the form of a SMS confirmation or key fob. You enter your username and password and then you enter the code messaged to your registered phone. Second factor authentication is an important step for login security, however, it burdens the user with additional work. Even if we understand why it's important, mankind is naturally lazy. Many sites allow users to opt out of repeated confirmations and many users readily select this time saving degradation of security. A further example method, may be to first validate with the device from which the authentication request is sent. Using a TPM or any other secure source of cryptographic key sets, a web service can ask the device to prove it is the same device it was before. This request can be transparent to the user (or further secured with a PIN) and provides a level of assurance whereby hassling the user for identity and authentication can often be bypassed.

[0019] A machine generated cryptographic proof tends to be far more reliable than a short username and eight character password, both of which are probably based on memorable facts attributed to the user. The user is best relegated to the job of protecting the device. Ten thousand years of evolution has trained people to protect valuable objects. Yet we find it hard to remember even a ten digit phone number. Devices, on the other hand, are purpose-built for blazingly fast math. If a user finds him or herself without a regularly used device, the service can fall back on user identification procedures. When it's not the common use case a user will be willing to accept more onerous identification procedures.

[0020] According to an example embodiment of the invention, the first step of leveraging device identity is enrollment. In one preferred embodiment, device enrollment may be enacted under the oversight of some other trusted entity. For example, enrollment of a phone could take place at the point of sale where binding between the end user and the device identity can be established with physical presence. However, in many use cases this level of person-to-device association is neither necessary nor desired. Device identity and attributes that could be considered Personally Identifying Information (PII) should not be inextricably linked. Basic device identity is purely anonymous. To reliably enroll a device we only need two things: A) The ability to generate a key pair that is locked to the device, and B) assurance of the provenance and quality of the device environment that provides this service. The latter is provided either by social engineering or supply chain crypto. While nothing is absolute, a device registered in the presence of a respected purveyor is likely to be a real device. It is important to the lasting reputation of that purveyor. Trust in a device that is keyed on the manufacturing floor and can be confirmed with the OEM certificate authority, likewise, is built on the reputation of that manufacturer.

[0021] According to some embodiments, enrollment involves establishing a uniqueness which can be queried but not spoofed. For this, the TPM (or similar hardware root of trust) may be used. The TPM chip generates a key pair and returns the public portion of the key to the client which in turn posts it to a server. A random id is generated and together the couplet is transacted into Namecoin (or similar block chain, or block chain method, devised to record named data.) Once ensconced in the block chain, the device record can be extended and modified with attributes such as the PCR quotes, associated Bitcoin accounts or other data. It is anticipated that large data objects will be referenced with a hash and URL in the block chain, rather than directly. The enrollment agent, in conjunction with the device, controls the Namecoin account that can update this record. However, one could imagine a scenario for self-enrolled devices where the enrollment agent is also the device. Once enrolled a service can access the public keys of the device to validate and encrypt communications and cryptographic assurance that the associated attributes emanated from the device.

[0022] In a trusted execution environment, the features of device identity are provided while further extending the ability to execute code in isolation from the rest of the system. Embodiments of this invention provide a Bitcoin Services Component that is packaged for deployment in a variety of TEE environments. This results in a couple of critical enhancements to the execution of a transaction: (1) Code is signed and authenticated by a third party trusted application manager so it can’t be tampered with. (2) Code is executed outside the host operating environment and is thus protected from malware. (3) Application data, beyond just keys, are never exposed outside of the TEE.

[0023] An enrolled device can build up a record of attributes that enable service providers to verify its state and context. Device attributes needn't include any PII to be useful. For example, a recent statement declaring a clean boot sequence can give a service provider some confidence that the machine is not compromised. Attributes that provide singular assertion of a fact can also be useful without divulging much PII, for example, the machine operator has been validated as over 21, or as a French citizen or member of an affinity club. In most cases, an interaction with a device is an opportunity to collect a statement of its boot integrity. This is nothing more than a collection of hashes that can be compared against the last boot statement. A machine that booted in a predictable way is believably more reliable than one who has changed BIOS or OS. In addition to PCR quotes, participating anti-virus software can deliver a statement that the machine was cleared as of the last scan.

[0024] In some embodiments, integration of the principles of Trusted Network Connect (TNC) would allow a full validation of an unknown client device prior to acceptance of a transaction. The client device being in a known good condition or state prior to the acceptance of a transaction is based on a third party's statement that the device is configured correctly. This type of verification addresses a broad range of cyber security controls that may be preferably required as part of any transaction processing system.

[0025] An exemplary embodiment is a computer-implemented method of verifying device integrity of a user device in a block chain communication network comprising in preparation for delivering an electronic transaction in the block chain network, implementing a device integrity verification process as part of the transaction including performing an internal validation of the integrity of the device execution environment from a root of trust in the user device; and requiring an electronic signature, such that a verification of the integrity of the signature is applied to the block chain transaction; wherein verification of the integrity of the signature is based on a determination of whether the execution environment of the device is in a known good condition including based on the integrity of the signature, allowing the transaction to proceed or requesting a remediation authority to verify that the electronic transaction as intended by the user is allowed to proceed even if it is determined that the execution environment of the device is not in a known good condition. In some embodiments verification of the integrity of the signature includes transmitting a root of trust instruction to the block chain network for processing, such that at least a portion of the block chain network responds by requiring multiple electronic signatures in order to accept the electronic transaction including creating within the execution environment of the device, an instruction from a root of trust in the user device; requiring a first electronic signature that corresponds to the root of trust instruction, such that a verification of the integrity of the signature is applied to the block chain transaction; and responding to the first electronic signature by verifying the integrity of the signature based on a determination of whether the execution environment of the device is in a known good condition including comparing the signature with a previously recorded reference value; if the signature matches the previously recorded reference value, then allowing the transaction to proceed; and if the signature does not match the previously recorded reference value, requesting a third party out of band process to verify that the electronic transaction as intended by the user is allowed to proceed even if it is determined that the execution environment of the device is not in a known good condition. In some embodiments, verifying the integrity of the signature includes the device providing the electronic signature based on a determination of whether the execution environment of the device is in a known good condition; allowing the transaction to proceed if the device provides the electronic signature; allowing the transaction as intended by the user to proceed even if it is determined that the execution environment of the device is not in a known good condition if the remediation authority provides the signature. Additionally, the out of band process may further include using an N or M cryptographic key function to confirm that at least one of an intent of the user meets predetermined requirements, or the device integrity meets predetermined requirements, or an additional process meets predetermined requirements. The reference value may be generated during a registration process performed by the owner of the device platform. The reference value may be generated based on a birth certificate assigned to the device, wherein the birth certificate is generated by the manufacturer or creator of the device, the manufacturer or creator of the execution environment of the device and/or the manufacturer or creator of an application on the device. The reference value may include a signature of at least one of the manufacturer or creator of the device, the manufacturer or creator of the execution environment of the device and/or the manufacturer or creator of an application on the device. The third party out of band process may return a token in response to the request to verify the transaction. Some embodiments may allow the electronic transaction to be completed within a certain period of time if the signature does not match the previously recorded reference value.

Some embodiments may verify that the intended electronic transaction is allowed to proceed even if it is determined that the execution environment of the device is not in a known good condition is based on a period of time between the registration of the reference value and the transaction and/or the amount of the transaction. Transactions above a threshold amount may be allowed to proceed if the period of time meets predetermined requirements. Allowing the transaction above a certain amount may be based on a minimum number of previously allowed transactions. Some embodiments may further comprise using a display device indicating to the user whether device integrity meets minimum predetermined requirements and further actions to be taken. Other embodiments may further include notification to a third party of the transaction, wherein in response to the notification, the third party records the transaction and a state of the device. The third party may record measurements associated with the device integrity for future analysis of the transaction. In addition, assuring the privacy of the record may include cryptographically obfuscating the record such that the record is made available only to authorized third parties. Another exemplary embodiment is a computer-implemented system of verifying device integrity of a user device in a block chain communication network comprising a block chain communication network; a user device in the block chain network; an electronic transaction in the block chain network; a device verification process implemented as a part of the transaction in preparation for delivery of the electronic transaction in a block chain network, the implementation further comprising an internal validation of the integrity of the device execution environment performed from a root of trust in the device; an electronic signature, such that a verification of the integrity of the signature is applied to the block chain transaction; wherein verification of the integrity of the signature is based on a determination of whether the execution environment of the device is in a known good condition including: based on the integrity of the signature, allowing the transaction to proceed or requesting a remediation authority to verify that the electronic transaction as intended by the user is allowed to proceed even if it is determined that the execution environment of the device is not in a known good condition.

BRIEF DESCRIPTION OF THE DRAWINGS

[0026] The foregoing will be apparent from the following more particular description of example embodiments of the invention, as illustrated in the accompanying drawings in which like reference characters refer to the same parts throughout the different views. The drawings are not necessarily to scale, emphasis instead being placed upon illustrating embodiments of the present invention.

[0027] FIG. 1A is an example digital processing environment in which embodiments of the present invention may be implemented.

[0028] FIG. 1B is a block diagram of any internal structure of a computer/computing node.

[0029] FIG. 2A is a block diagram showing an example device authentication system according to the invention.

[0030] FIG. 2B is a diagram showing an example device authentication system according to the invention.

[0031] FIG. 2C is a diagram of the components of an embodiment of the invention.

[0032] FIG. 2D is a diagram of the Authentication System Adaptor and its outward and inward looking interfaces.

[0033] FIG. 3A is a diagram of the sequence of packaging and delivering an instruction by the Encoder.

[0034] FIG. 3B is a diagram of the device enrollment process according to an embodiment of the invention.

DETAILED DESCRIPTION

[0035] A description of example embodiments of the invention follows.

[0036] Embodiments of the present invention are systems and methods for attesting to device health prior to engaging in electronic transactions.

[0037] Block chain transactions do not have verification or cyber security controls on an unknown device performing the transactions. Therefore, a full validation of an unknown client device prior to acceptance of a block chain transaction would provide further security for block chain transactions.

[0038] Example embodiments may be founded on the principles of the Trusted Network Connect (TNC) standards under which the integrity of a device may be verified prior to actual enablement of the connection to a network switch. According to TNC, a device performs a series of measurements that are securely stored on the device. The measurements typically would include validation of the BIOS image, the operating system (OS) and any applications that need to be verified that they have not been altered. Upon connection to the network, the switch would perform a validation process verifying that the measurement data matches a reference value that was computed when the device was either previously connected or in a current known good condition or state. The Trusted Execution Environment (TEE) is also capable of self-measurement processes and remote attestation of the health of the device. In some preferred embodiments, the TNC system is based on the Trusted Computing Group (TCG) standards and typically the Trusted Platform Module (TPM) chip is integrated.

[0039] In some embodiments, automation of full device integrity verification is provided as part of a block chain transaction. In order to provide a validation of the device integrity, a device that is performing a block chain instruction would perform an internal validation of the integrity of the execution environment from a root of trust in the device at the initialization of the block chain transaction. The device would, with or without Human input create an instruction within the measured environment. This instruction would then be sent to the block chain network for processing. The block chain network will require multiple signatures to accept the transaction. The first signature would be the created root instruction itself that would have the verification of the signature applied to the transaction. The network then verifies the integrity signature of the execution environment by comparing it with a previously recorded Reference Value. If the signature matches the Reference Value the transaction is allowed to proceed. If the signature and Reference Value do not match then the system will require a third out of band process to be completed that would verify that the transaction intended is allowed to proceed even if the execution environment is not in a known good condition. Because, block chain transactions do not have any verification or cyber security controls on an unknown device performing a transaction, embodiments of the present invention would allow a full validation of an unknown client device being in a known good condition according to a third party's statement that the device is configured correctly prior to the acceptance of a transaction. Some embodiments of the present invention, therefore, can address a broad range of cyber security controls that should be required as part of any block chain transaction processing system.

[0040] Digital Processing Environment

[0041] An example implementation of a system according to the invention for attesting to device health prior to engaging in transactions 100 may be implemented in a software, firmware, or hardware environment. FIG. 1A illustrates one such example digital processing environment in which embodiments of the present invention may be implemented. Client computers/devices 150 and server computers/devices 160 (or a cloud network 170) provide processing, storage, and input/output devices executing application programs and the like.

[0042] Client computers/devices 150 may be linked directly or through communications network 170 to other computing devices, including other client computers/devices 150 and server computer/devices 160. The communication network 170 can be part of a wireless or wired network, remote access network, a global network (i.e. Internet), a worldwide collection of computers, local area or wide area networks, and gateways, routers, and switches that currently use a variety of protocols (e.g. TCP/IP, Bluetooth®, RTM, etc.) to communicate with one another. The communication network 170 may also be a virtual private network (VPN) or an out-of-band network or both. The communication network 170 may take a variety of forms, including, but not limited to, a data network, voice network (e.g. land-line, mobile, etc.), audio network, video network, satellite network, radio network, and pager network. Other electronic device/computer networks architectures are also suitable.

[0043] Server computers 160 may be configured to provide a user device authentication system 100 which communicates with authenticators to confirm a requestor's identity prior to allowing the requestor to access resources protected by the authentication system. The server computers may not be separate server computers but part of cloud network 170.

[0044] FIG. 1B is a block diagram of any internal structure of a computer/computing node (e.g., client processor/ device 150 or server computers 160) in the processing environment of FIG. 1A, which may be used to facilitate displaying audio, image, video or data signal information. Each computer 150, 160 in FIG. 1B contains a system bus 110, where a bus is a set of actual or virtual hardware lines used for data transfer among the components of a computer or processing system. The system bus 110 is essentially a shared conduit that connects different elements of a computer system (e.g., processor, disk storage, memory, input/output ports, etc.) that enables the transfer of data between elements.

[0045] Attached to the system bus 110 is an I/O device interface 111 for connecting various input and output devices (e.g., keyboard, mouse, touch screen interface, displays, printers, speakers, audio inputs and outputs, video inputs and outputs, microphone jacks, etc.) to the computer 150, 160. A network interface 113 allows the computer to connect to various other devices attached to a network (for example the network illustrated at 170 of FIG. 1A). Memory 114 provides volatile storage for computer software instructions 115 and data 116 used to implement software implementations of device integrity attestation and authentication components of some embodiments of the present invention. Such device integrity attestation and authentication software components 115, 116 of the user authentication system 100 (e.g. encoder 210, Trusted Execution Environment (TEE) applet 208, authentication site 206 of FIG. 2A) described herein may be configured using any programming language, including any high-level, object-oriented programming language, such as Python.

[0046] In an example mobile implementation, a mobile agent implementation of the invention may be provided. A client server environment can be used to enable mobile security services using the server 190. It can use, for example, the XMPP protocol to tether a device authentication engine/agent 115 on the device 150 to a server 160. The server 160 can then issue commands to the mobile phone on request. The mobile user interface framework to access certain components of the system 100 may be based on XHP, Javelin and WURFL. In another example mobile implementation for OS X and iOS operating systems and their respective APIs, Cocoa and Cocoa Touch may be used to implement the client side components 115 using Objective-C or any other high-level programming language that adds Smalltalk-style messaging to the C programming language.

[0047] The system may also include instances of server processes on the server computers 160 that may comprise an authentication (or attestation) engine 240 (FIG. 2), which allow registering a user, selecting authenticators/attesters for confirming a requestor is a registered user, communicating with the authentications in regards to confirming a requestor's identity, and executing algorithms, such as statistical algorithms to compute confidence scores, to allow or deny the requestor access to resources protected by the system.

[0048] Disk storage 117 provides non-volatile storage for computer software instructions 115 (equivalently "OS program") and data 116 used to implement embodiments of the system 100. The system may include disk storage accessible to the server computer 160. The server computer can maintain secure access to records related to the authentication of users registered with the system 100. Central processor unit 112 is also attached to the system bus 110 and provides for the execution of computer instructions.

[0049] In an example embodiment, the processor routines 115 and data 116 are computer program products. For example, if aspects of the authentication system 100 may include both server side and client side components.

[0050] In an example embodiment, authenticators/attesters may be contacted via instant messaging applications, video conferencing systems, VOIP systems, email systems, etc., all of which may be implemented, at least in part, in software 115, 116. In another example embodiment, the authentication engine/agent may be implemented as an application program interface (API), executable software component, or integrated component of the OS configured to authenticate users on a Trusted Platform Module (TPM) executing on a computing device 150.

[0051] Software implementations 115, 116 may be implemented as a computer readable medium capable of being stored on a storage device 117, which provides at least a portion of the software instructions for the user authentication system 100. Executing instances of respective software components of the user authentication system 100, such as instances of the authentication engine, may be implemented as computer program products 115, and can be installed by any suitable software installation procedure, as is well known in the art. In another embodiment, at least a portion of the system software instructions 115 may be downloaded over a cable, communication and/or wireless connection via, for example, a browser SSL session or through an app (whether executed from a mobile or other computing device). In other embodiments, the system 100 software components 115, may be implemented as a computer program propagated signal product embodied on a propagated signal on a propagation medium (e.g. a radio wave, an infrared wave, a laser wave, a sound wave, or an electrical wave propagated over a global network such as the Internet, or other networks. Such carrier medium or signal provides at least a portion of the software instructions for the present user device authentication system 100 of FIG. 2A.

[0052] Certain example embodiments of the invention are based on the premise that online services may be significantly enhanced when a device can be trusted to be what it says it is and to execute instructions exactly as asked. A service provider generally has confidence in its servers because they are under administrative control and usually protected physically. However, nearly all of the service provider’s services are delivered to users through devices the service provider knows very little about and over which it rarely exerts any control.

[0053] Through the use of Trusted Execution technology, certain ineventive embodiments are able to provide a service provider with an oasis of trust in the unknown world of consumer devices. Basic capabilities such as "sign this", or "decrypt this" are executed outside the murky world of the main OS. Keys can be generated and applied without ever being exposed in memory and can be attested to through a chain of endorsements traced back to the device manufacturer.

[0054] Certain aspects of the invention enable trust in devices. Some embodiments operate on the fundamental premise that a reliable relationship with a device can make for a much safer, easier and stronger relationship with an end user. Achieving this requires knowing with confidence that a device involved in a current transaction is the same device it was in previous transactions. It also requires assurance that a device will not leak protected information if it is requested to perform sensitive operations such as decryption or signing.

[0055] One example preferred embodiment includes device code executed in the Trusted Execution Environment (TEE). The TEE preferably is a hardware environment that runs small applets outside the main OS. This protects sensitive code and data from malware or snooping with purpose-built hardware governed by an ecosystem of endorsements, beginning with the device manufacturer.

[0056] Device Integrity Attestation/Authentication - Some Example Embodiments

[0057] FIG. 2A is a block diagram showing an example device authentication system according to the invention, with components 200. With these system components 200, web developers and app developers can make use of hardened encryption and identity keys in endpoint User Devices 205 through an application program interface (API). In addition, further services may be provided built on these system components 200 for device management, backup, attestation, etc. To support this system, the registration of identity keys and a set of device management services for attestation, backup and device grouping, are managed.

[0058] In a preferred example embodiment, it would be the intent of the system not to maintain mission critical data as in conventional approaches, but rather to provide a platform for seamless yet very secure connections between Service Providers 204 and User Devices 205. On one end of the system is the Encoder 210 which prepares an instruction for a User Device 205 and at the other is the Device Rivet which is the Trusted Execution Environment (TEE) applet 208 that can act on that instruction. A Protocol according to an embodiment of the invention defines how these instructions and replies are constructed.

[0059] The Device Rivet or TEE applet 208 preferably embodies the innovative binding between the physical and digital works. The Device Rivet or TEE applet 208 locks features of identity, transaction and attestation to the hardware of the Device 205.

[0060] The system 200, according to an embodiment of the invention shown in FIG. 2B, may use a secure socket to maintain a persistent connection with all devices. This channel is used for pairing and other administrative functions. Library code 209 may be provided to service providers for simplifying the construction and signing of an instruction. This Library 209, for example, could be implemented in a programming language, such as an object-oriented, high-level programming language with dynamic semantics like Python.

[0061] In one example preferred embodiment, the TEE may be implemented as a mobile phone hardware security chip separate execution environment that runs alongside the Rich Operating System and provides security services to that rich environment. The TEE offers an execution space that provides a higher level of security than a Rich OS. In another example embodiment, the TEE may be implemented as a virtual machine. While not as secure as a Secure Element (SE) (aka SIM), the security offered by the TEE is sufficient for some / many applications. In this way, the TEE can deliver a balance allowing for greater security than a Rich OS environment with considerably lower cost than an SE.

[0062] The Ring Manager 212 can be implemented as a service provided to end-users for managing collections (or Rings) of User Devices 205. Devices 205 may be grouped into a single identity and used to backup and endorse each other. Rings may be associated with other rings to create a network of devices. In some preferred embodiments, the rings are a collection of individual device public keys (as opposed to a new key). If there are not many shared devices in the environment, preferably the list of devices preferably may short because of the potential for increased computational and bandwidth resources may expended and introduce a time cost in order to encrypt a message with all of the public keys on a device list.

[0063] In a non-preferred example embodiment, a ring may be implemented as a shared private key on top of the unique private key of the Device 205. It should be noted, however, it is not typical to share a “private key”, nor would it be desirable to have a long-lived shared symmetric key.

[0064] One aspect of the system according to an embodiment of the invention enrolls a device and equips it with a service provider's keys. Inventive API's enable secure execution of a number of sensitive device-side transactions, including: getting a reliable and anonymous device id - on request, an embodiment of the invention will generate a signing key for a device. The public key is hashed into a string that can be used to identify and communicate with a device. The private key remains locked in the hardware and can only be applied on behalf of the service that requested the ID; getting a device to sign something - the private key of the device identity can be used to sign things proving that this particular device was involved. The signing ceremony is executed in secure hardware such that the key is never exposed to normal processing environment of the device; getting a device to encrypt something - an encryption key can be generated on request and applied to any blob of data. Encryption and decryption is triggered locally and takes place within the secure execution environment so as to protect the key; creating a Bitcoin account - the device can be asked to generate a new Bitcoin account using the random number generator (RNG) built into the TEE; signing a Bitcoin transaction - the device can apply its private Bitcoin account key to sign a transaction and then return it to the service provider; securing confirmation - newer TEE environments support trusted display and input in addition to trusted execution. Trusted display enables a simple confirmation message, such as "confirm transaction amount," to be presented to an end user; joining devices to share and backup identities - most users have several devices. Certain embodiments of the invention enable multiple devices to be bound into a *ring* so they can interchangeably present themselves to a service provider on behalf of the user.

[0065] A Service Provider calls a Third Party Agent/Process to create hardware keys in a device. Different types of keys are available depending on the purpose, such as for crypto-coins or data encryption. Hardware keys are governed by simple usage rules established during creation. For example, a key may require that usage requests are signed by the Service Provider that created the key, or that the user confirms access through the Trusted User Interface (TUI).

[0066] A Device Rivet 208 will only respond to an instruction from a Service Provider 204 that has been "paired" with the Device 205. The Authentication Web Site 206 conducts the pairing ceremony as it is able to confirm the integrity and identity of both device and the service provider. When a Device 205 is paired it acquires the public key of the Service Provider 204, while the Service Provider gets a uniquely generated identity and public key for the Device 205.

[0067] While the Third Party Agent/Process supports local calls, ideally all instructions are signed by the Service Provider 204. This protects a device key from being applied by a rogue application. An Encoder 210 is provided to help prepare and sign device instructions on the application server.

[0068] There is a class of apps that benefit greatly from strong assurance of their origin and opaque separation from the execution of other apps. This is known as a Trusted Execution Environment or TEE. Unlike an app running on the primary OS and memory stack, an app running in a TEE has access to cryptographic primitives that can be exercised without snooping by the OS. On certain platforms, the app also has direct access to user input and display to ensure a private interaction with the operator of the device. While the technology has been pursued for well over a decade, it is only recently that devices with support for a TEE have become available. For example, Intel began delivery of commercial solutions in 2011 and Trustonic, an ARM joint venture, was launched in 2013.

[0069] Deploying an applet into a TEE is akin to delivering a dedicated hardware device. Execution and data are cryptographically isolated from any other function of the host. While most applications of Trusted Execution technology have been concerned with enterprise security or DRM, an embodiment of the invention instead provides an applet that is focused on the needs of common web services. Crypto currencies such as Bitcoin have highlighted the need for consumer key security.

[0070] An embodiment of the invention provides a native API that translates calls into a secure environment. While different TEE environments follow very different architectures, the API of an embodiment of the invention is designed to present a uniform interface to the application.

As with all TEE applets, TEE applets according to embodiments of the invention cannot be installed and initialized without a Trusted Application Manager, or TAM. The TAM plays a role akin to a certification authority (CA). A TAM secures a relationship with a device manufacturer and also signs all applets that may be loaded into the device. In this way the TAM expresses assurance about the provenance and integrity of both the applet and the TEE.

[0071] Device Integrity Attestation

[0072] Embodiments of the invention provide device integrity attestation by automating the assurance of device integrity against a known state as a signatory on a block chain transaction. The system implemented by an embodiment of the invention is comprised of the several components shown in FIG. 2C. A Device Adapter 220 is a software service running on an endpoint device that provides an interface to a Service Provider 204 application and integrates with the Device TEE 208. The Trusted Execution Environment (TEE - sometimes TrEE) is a mobile phone hardware security chip separate execution environment that runs alongside the Rich OS and provides security services to that rich environment. The TEE offers an execution space that provides a higher level of security than a Rich OS; though not as secure as a Secure Element (SE) (aka SIM), the security offered by the TEE is sufficient for some / many applications. In this way, the TEE delivers a balance allowing for greater security than a Rich OS environment with considerably lower cost than an SE. Another component, the Device TEE 208 is a software program that executes in a hardware secured TEE. The Device TEE 208 is specially designed to execute cryptographic functions without compromise from malware or even the device operator. Another component, the Device Registrar 221 is a service that registers a device into the block chain 222. A block chain 222 is used both to store device registration and attributes and to execute transactions. There may be different block chains. Another supporting component is a Service Provider 204 which is the application seeking to conduct a transaction with a device. OEM (Original Equipment Manufacturer) 223 is the entity that built the device and/or a Trusted Application Manager (TAM) authorized to cryptographically vouch for the provenance of the device.

[0073] According to an embodiment of the invention, when the Device Adapter 221 shown in FIG. 2C software runs for the first time it will ask the Device TEE 208 to generate a public/private key pair. The public key is signed by an endorsement key established during device manufacturing. This signed public key is sent to the Device Registrar 221 and validated with the OEM 223. Registration may involve confirmation from the device operator. Registration may involve endorsement at the point of sale in the presence of a clerk. The Registrar may ask the device for a Device Measurement Record which includes one or more of the following: a composite value of the Platform Configuration Registers (PCR's) generated by the boot process, BIOS Version, OS Version, GPS Location. This data is signed by the device private key. It is further signed by the Registrar. The resulting data set becomes the gold reference or Reference Value for future integrity checks. Confirmation from the device operator may be required in collecting the gold reference or Reference Value. This data set is posted into a public cryptographic ledger. The public record established cryptographic proof of the time of registration along with the endorsement of the registrar. The registration may further include attribute data, such as location or company name or device make/model. The registration may reference a signed document that sets out the policy terms of the registrar at the time of registration. The Device Registrar 221, or another trusted integrity server, creates a block chain account key (a public/private key pair) that can be referenced as a signatory in a multi-signature transaction on the block chain. A signatory the value represented in the block chain transaction cannot be spent or transferred unless co-signed by the Registrar.

[0074] To sign a transaction the integrity server expects a recent measurement from the device. This measurement may be requested directly of the Device Adaptor or fetched by the server through a persistent sockets connection with the device. The current measurement is compared against the gold measurement or Reference Value in the block chain. If the measurements match the transaction is signed. If the measurements match but the recent measurement is older than a specified time window, the request is rejected. If the measurements do not match, the request is rejected. If there is a rejection, the transaction may have been prepared with another manual signatory that can be asked to override the rejection. If the measurements do not match, the device may be put through a registration renewal where a new measurement is gathered. Every time a measurement matches, the device registration record can be updated with a success count. The integrity server may be given policy rules that will accept a measurement which doesn't match if the problem is not deemed severe in light of other matching measurements or attributes.

[0075] A system, according to an embodiment of the invention, may be implemented with a collection of trusted devices rather than an integrity server to do the work of matching measurements and signing the transaction. The system may match integrity measurements directly during transaction processing using features built into a smart block chain system such as that being developed by Ethereum.

[0076] Device Integrity Attestation - Authentication Web Site

[0077] In an example embodiment, Authentication Web Site 206 may be a JSON API written in Python, which uses the Third Party Agent/Process private key to enroll the identity keys of Devices 205 and Service Providers 204. During enrollment, the public key of the User Device 205 or Service Provider 204 is recorded by the TEE applet 208. Enrollment enables the TEE applet 208 to pair a Device 205 with a Service Provider 204. The result of pairing is that a User Device 205 has a service public key, endorsed by a Third Party Agent/Process and can therefore respond to Service Provider 204 instructions.

[0078] The Protocol according to an embodiment of the invention specifies the structure of an instruction and the signing/encryption that must be applied for the Device 205 to accept the instruction. The instruction itself may, for instance, be prepared as a C structure that contains the instruction code, version data and payload. The entire structure preferably is signed by the service provider key and delivered to the device TEE applet 208 by calling a device local command.

[0079] Preferably, every User Device 205 should present unique identity credentials. Devices may join a ring so as to act as a singular entity. In one embodiment, a Device 205 can support group ID's that are locally stored as a list, but publicly translate into cross-platform authentication. The TEE Adapter 216 may be configured as the interface between the Device Rivet/TEE applet 208 bolted into the TEE and the outside world of partner apps and online services. In implementation, it can manifest in one or more diverse forms, which would be at least partially dictated by the basic capabilities across devices, hardware support and OS architecture.

[0080] Device Integrity Attestation - Authentication System Adaptor

The Authentication System Adaptor 214 is composed of outward and inward looking interfaces as shown in FIG. 2D. The inward looking interface, the TEE Adapter 216, handles proprietary communications with the Device Rivet 208. The Host Adaptor 217 is provided to expose services to third-party applications. The Host Adaptor 217 presents the interface of the Authentication System Adaptor 214 through different local contexts, such as browsers or system services. Multiple realizations for diverse contexts are anticipated though initially this may be an Android service and a windows com process. The Socket Adaptor 215 connects the client environment Authentication Web Site 206. The TEE Adaptor 216 component is the proprietary glue that pipes commands into the Device Rivet 208. In an Android implementation the Authentication System Adaptor 214 may manifest as an Android NDK service app and may be configured to launch at boot. The Authentication System Adaptor 214 prepares message buffers that are piped to the Device Rivet 208 and then synchronously awaits notification of a response event. The Host Adaptor 217 is primarily there to isolate the TEE Adapter 216 from the host environment. The Host Adaptor 217 operates in a potentially hostile environment. There will therefore typically be limited assurance that the client has not been compromised. The Host Adaptor's role is therefore primarily to facilitate easy access to the Device Rivet 208. Instructions from a Service Provider 204 intended for the Device Rivet 208 will be signed by the Service Provider 204 and then passed through to the TEE Adapter 216 and Device Rivet 208.

[0081] First Service Provider Registered to a Device

[0082] According to an example embodiment, the Authentication Web Site 206 is the first service provider registered to a Device 205. The Authentication Web Site 206 has the special capability of being able to pair additional service providers with that Device 205. Communications with the Authentication Web Site 206 may be handled through the web API and should be authenticated. In one example, this is implemented with an API key. In a preferred example embodiment, this is implemented using an SSL key swap. In some embodiments, all requests will be signed.

[0083] The relationship with devices may be dependent on being able to sign instructions with the private key. Such a private key is highly sensitive and is protected. Preferably, the private key is encased in an HSM.

[0084] In some embodiments, multiple keys are used, such that if one is compromised the whole system is not lost. This should, for example, should make it more difficult for an attacker to know which devices are connected with a compromised key. Furthermore, the system 200 is preferably in near constant contact with all Devices 205 through the Socket Adapter 215 shown in FIG. 2C, which can facilitate frequent rotation of the keys.
The Authentication Web Site 206 may comprise several sub-components. A Device ID is the unique identifier, in a UUID, assigned to a device by the Authentication Web Site 206 or other Registration Agent. An ephemeral pointer, Device Pointer, may be provided to a device 150 that can be requested by any local application. The Device Pointer can identify a current socket session to the Authentication Web Site 206 and therefore can be used to establish a device communication channel and to look up the permanent identifier, the Device ID. The root of a device registration includes a unique, anonymous identifier, a registration date, a public key paired to a private key held in the device hardware and an endorsement signature from the Registration Agent. This information is recorded in the Device Registration Record. The TEE applet 208 embodies the binding between the physical and digital works. The Device Rivet 209 locks features of identity, transaction and attestation to hardware.

[0085] Protocol for Processing Instructions

[0086] The counterpart to the Device Rivet 209 is the Encoder 210. The Encoder 210 prepares a command to be executed by a specific device which is signed and/or encrypted by the Service Provider 204. The Service Provider public keys are preloaded into the device during a pairing process conducted by

Authentication Web Site 206. This allows the Device Rivet 209 to validate the origin of the request, and if needed decrypt the contents of the instruction.
The sequence of packaging and delivering an instruction is shown in FIG. 3A. The Service Provider 204 generates an Instruction Record with the help of the Encoder 210 libraries. The instruction includes the type, the target device and payload. The instruction may be encoded with the device key and must be signed by the service provider key. The device key is fetched from the Authentication Web Site 206, or directly from the block chain, by looking up the Device Registration Record.

[0087] Protocol for Enrolling the Device

[0088] Device enrollment or creation of a birth certificate for a device on the block chain is essential to example embodiments of the invention. The enrollment process, shown in FIG. 3B, must be hassle free or even transparent to the user. Ideally, a fully reputable Device ID would include personalization of the relationship between a device and a user with a PIN or other memory test; as well as legal binding between the user and the device, for example, by registering the device in presence of a sales clerk. It would look up the endorsement keys of the OEM that manufactured the device to ensure provenance. It also might include training on the purpose, power and anonymity of device registration. We can start with just creating the ID transparently. Because of this variability in the context of the registration, the Registration Agent should record the context of enrollment to ensure that trust is being extended where it's due. For example, testing an OEM endorsement key makes it vastly more certain that the Device Rivet is operating in a proper TEE.

[0089] In an example embodiment shown in FIG. 2C, when the Device Adapter 220 software runs for the first time it will ask the Device TEE 208 to generate a public/private key pair. The public key is signed by an endorsement key established during device manufacturing. This signed public key is sent to the Device Registrar 221 and validated with the OEM 223. Registration may involve confirmation from the device operator or registration may involve endorsement at the point of sale in the presence of a clerk. The Registrar 221 will ask the device for a Device Measurement Record which includes one or more of the following: a composite value of the Platform Configuration Registers (PCR's) generated by the boot process, BIOS Version, OS Version, GPS Location, BIOS identifier, a network interface identifier, attributes about the Device, such as number of files, size of files, directories, indexes and data/search tree structures, processor identifying number of the Device, or other such information. This data is signed by the device private key and may be further signed by the Registrar 221. The resulting data set becomes the gold reference for future integrity checks. Confirmation from the device operator may be required in collecting the gold reference. This data set is posted into a public cryptographic ledger, such as Namecoin. The public record established cryptographic proof of the time of registration along with the endorsement of the registrar. The registration may further include other attribute data, such as location or company name or device make/model. The registration may reference a signed document that sets out the policy terms of the registrar at the time of registration. The Device Registrar 221, or another trusted integrity server, creates a block chain account key (a public/private key pair) that can be referenced as a signatory in a multisig transaction on the block chain. A signatory value represented in the block chain transaction cannot be spent/transferred unless co-signed by the Registrar 221. To sign a transaction the integrity server expects a recent measurement from the device. This measurement may be requested directly of the device adapter or fetched by the server through a persistent sockets connection with the device. The current measurement is compared against the gold measurement in the block chain. If the measurements match the transaction is signed, if the measurements match but the recent measurement is older than a specified time window, the request is rejected. If the measurements do not match the request is rejected. If there is a rejection, the transaction may have been prepared with another manual signatory that can be asked to override the rejection. If the measurements do not match the device may be put through a registration renewal where a new measurement is gathered. Every time a measurement matches, the device registration record can be updated with a success count. The integrity server may be given policy rules that will accept a measurement which does not match if the problem is not deemed severe in light of other matching measurements or attributes. This system may be implemented with a collection of trusted devices rather than an integrity server to do the work of matching measurements and signing the transaction. This system may match integrity measurements directly during transaction processing using features built into a smart block chain system such as that being developed by Ethereum.

[0090] Birth Certificate for a Device on the Block Chain

[0091] An embodiment may be a method for creating a birth certificate for a user device in a block chain communication network comprising: establishing a device identity for the user device by generating a public/private key pair that is locked to the user device; signing of the public key of the device by an endorsement key established during manufacturing or creation of the device, manufacturing or creation of the execution environment of the device and/or manufacturing or creation of an application on the device; and enrolling the device with a trusted third party including: requesting and obtaining the generated public key from the device; requesting and obtaining a device measurement record of the device containing attributes related to the device Platform Configuration Registers (PCR), BIOS, OS and/or GPS; endorsing of the device measurement record by the third party and the device; and registering the device into the block chain including posting the endorsed device measurement record into a public cryptographic ledger; and creating a block chain account key pair that can be referenced as a signatory in a multi signature transaction on the block chain. In some embodiments the method may include enrolling the device with a third party is at the request of the first service provider seeking to pair with the device. In some embodiments, enrolling the device may be provided as a service. Endorsing of the device measurement record by the device may include signing of the record by the device private key. Endorsing of the device measurement record by the third party may be provided as a service. The registration may further include signing of a document that sets out the policy terms of the registration provider at the time of registration. The public cryptographic ledger may be Namecoin. The endorsed device measurement record may establish a Reference Value for transactions between a service provider and the device. Additionally, confirmation by the device operator is required to obtain the device measurement record of the device attributes from the device. The device attributes may further include location, company name and/or device make/model. Further, the transaction between a service provider and the device may require the device to generate and provide a device measurement record that is compared to the established Reference Value for the device. In other embodiments, the transaction is allowed if the comparison results in a match or the transaction is rejected if the comparison results in no match or the transaction is rejected if the comparison results in a match and the record provided by the device is older than a specified time window or the device is required to re-create its birth certificate if the comparison results in no match. Additionally, registering the device into the block chain may further include creating a device registration record that is updated with a success count if the comparison results in a match. The comparison may be implemented by a collection of trusted devices. The entity performing the comparison may be independent of the entity performing the registration.

[0092] Another embodiment may be a system comprising a block chain communication network; a user device in the block chain network; a trusted third party; and a system for creating a birth certificate for the user device, said system configured to establish a device identity for the user device by generating a public/private key pair that is locked to the user device; sign the public key of the device using an endorsement key established during manufacturing or creation of the device, manufacturing or creation of the execution environment of the device and/or manufacturing or creation of an application on the device; and enroll the device with the trusted third party by: requesting and obtaining the generated public key from the device; requesting and obtaining a device measurement record of the device containing attributes related to the device Platform Configuration Registers (PCR), BIOS, OS and/or GPS; endorsing of the device measurement record by the third party and the device; and registering the device into the block chain by posting the endorsed device measurement record into a public cryptographic ledger; and creating a block chain account key pair that can be referenced as a signatory in a multi signature transaction on the block chain.

[0093] Using Transactions on the Block Chain to Accumulate Ownership Rights

[0094] A bitcoin Wallet functions similarly to a bank account and can be used to receive and store bitcoins as well as transfer them to others in the form of electronic transaction in the Bitcoin block chain. A bitcoin address is a unique identifier that allows a user to receive bitcoins. Bitcoins are transferred by sending them to a bitcoin address. The transactions in the bitcoin block chain are usually free. However, transactions that send and receive bitcoins using a large number of addresses will usually incur a transaction fee. A Wallet stores the private keys so that the user can access bitcoin addresses.

[0095] Systems and methods may be provided whereby a transaction on the block chain accumulates or achieves an ownership right.

[0096] A service may be provided whereby a bitcoin transaction accumulates to a new license right. This would be done by integrating a smart contract with attribute information in the transaction record that would identify the chain of transactions that accumulate to a right. Ultimately this right would be bound to the original Wallet address. Every time a specific item is purchased it would incorporate the last transaction as part of the attribute data of the current transaction assuring that the accumulation of transactions could be quickly and efficiently verified by reading the information on the block chain. The act of performing many small transactions on the block chain would enable an account to easily accumulate to an ownership right or a replay right. Once a specific level is reached, the accumulation would stop and a persistent right would be written to the block chain.

[0097] Some embodiments may include systems and methods for attesting to device health prior to engaging in electronic transactions.

[0098] This would be done by integrating a smart contract with attribute information in the transaction record that would identify the chain of transactions that accumulate to a right. Ultimately this right would be bound to the original Wallet address. Every time a specific item is purchased it would incorporate the last transaction as part of the attribute data of the current transaction assuring that the accumulation of transactions could be quickly and efficiently verified by reading the information on the block chain. The act of performing many small transactions on the block chain would enable an account to easily accumulate to an ownership right or a replay right. Once a specific level is reached, the accumulation would stop and a persistent right would be written to the block chain.

[0099] A system for may be provided for accumulating a value attached to transactions in a block chain communication network associated with a bitcoin account, the system comprising a block chain communication network; an electronic transaction in the block chain network; a bitcoin account; a transaction record associated with the bitcoin account; a transaction interrogation process implemented as a part of executing the electronic transaction in a block chain network. The implementation may further comprise a checking of the transaction record for the existence of a previous transaction associated with the account; and based on the existence of a previous transaction: obtain an accumulated value attached to the previous transaction; increment the obtained accumulated value; attach the incremented accumulated value to the transaction in the transaction record; and apply the incremented accumulated value to the transaction.

[00100] The implementation of the transaction interrogation process may further comprise setting a plurality of charges incurred for executing the electronic transaction to zero and indicating the achievement of a Right associated with the account, based on the incremented accumulated value reaching or exceeding a predetermined maximum accumulated transaction value.

[00101] The implementation of the transaction interrogation process may further comprise creating a new transaction record associated with the account; and storing an indication of the achieved Right in the newly created transaction record.

[00102] The electronic transaction may be associated with a specific Item, the transactions in the transaction record associated with the account form a chain with cryptographic assurance and the implementation of the transaction interrogation process may further comprise: allowing a user to query the last transaction recorded in the transaction record associated with the account; and calculating a level of expenditure for the specific Item based on cryptographic assurance of the formed chain.

[00103] Applying the accumulated value to the transaction may include associating the achieved Right with a cryptographic key; storing the key in a tamper resistant storage; obtaining a set of transactions contributing to the accumulated value associated with the achieved Right; and verifying the set of transactions prior to applying the accumulated value to the transaction.

[00104] In some systems, the set of transactions must be completed within a specific period of time in order to contribute to the achievement of the Right. The achieved Right expires after a specific period of time and/or expires based on the lack of use of the Right. The achieved Right is used as part of a multiple signature transaction to enable the purchase of additional transactions requiring an indication of the achieved Right.

[00105] In some systems, the transaction is associated with a single Item and involves two achieved Rights and the accumulated values associated with the Rights are cryptograhically merged to result in a single accumulated value.

[00106] Assured Computer Instructions to Cloud Services and Peer Services

[00107] The current state of computing is based on an authentication model in which devices connect to a cloud service like Twitter and then assume that the follow-on data is correct. Encrypted transport is commonly used and the assurance model is based on assuring the whole computer that sends the data. Technologies like anti-virus and integrity validation are provided for the host system. An assumption is made that the complex system is okay and to trust the critical data delivered.

[00108] Authentication may be augmented with assured computer instructions that are formed within the local device from both remote sources to assure these instructions are correct and to then deliver these instruction to remote services for processing. The system may collect data from user input, device input, remote system input and then provide a secure mechanism for the user to confirm this is the intended transaction to be performed. The cloud service receives this assured instruction and verifies that the elements of the transaction are correct. The verification process may also impose local or remote policies that are verified prior to the transaction being accepted for processing. The resulting data can then be logged.

[00109] In a general purpose computing device, typically, authentication is used to connect to critical services. Even with strong authentication there is no assurance that the information sent to the cloud is the information the user intends. Malware can find many ways to alter the data and result in the theft or compromise of sensitive data. The purpose of this invention is to collect a number of sources of both local and remote data to assure that the information provided is the data that is intended. Certain data could also be locally masked to assure a process has been completed but the detailed private information remains masked. Services can then verify the transactions are intended and incorporate a number of additional process steps internally and externally that are controlled by the user. This can assure logging and additional verification to assure the transaction is correct. This can be used in financial systems but also to control the internet of things from door locks to medical devices.

[00110] In some systems, a secure sub system is used for assembling a secure instruction for delivery to another computer system. The secure sub system collects and appends additional information such as time, location, identity, compliance or other critical data locally or remotely and provide the user a mechanism to securely confirm the instruction prior to the instruction being signed and then sent.

[00111] In some systems, when the protected instruction is received, it is verified prior to being processed. Verification can be done locally or remotely and may include additional user verification, confirmation or signature from logging systems, other critical process steps, location or time.

[00112] In some systems, local data could be tokenized to protect privacy. For example, the users phone number could be used to say they are a specific provider's customer and in good standing but all that is passed on is the good standing status and not the users name or phone number. This is done by contacting the provider locally and having the confirmation data include a provider transaction identity that can be remotely verified.

[00113] Some systems may leverage the local attestation data to assure the isolated execution environment can be prove that it is in a known condition at the time of the transaction.

[00114] Systems may be configured with a logic script that is cryptographically assured to provide the policy required for a specific transaction. The script validation may be included as part of the transaction verification data.

[00115] Systems may include local or remote approvals prior to the transaction being released (i.e. multi signal on the client side). The systems may receive real time data that is locally assured and then modified so the instruction is a delta to a real time state, for example, to increase speed of a pump. In some systems, the verifying device assures that the transaction came from a known source that meets the minimum number of parameters. In other systems, the receiving device additionally verifies local or remote information.

[00116] While this invention has been particularly shown and described with references to example embodiments thereof, it will be understood by those skilled in the art that various changes in form and details may be made therein without departing from the scope of the invention encompassed by the appended claims.

APPENDIX

1. Component Specification

2. System Overview

Rivetz enables web developers and app developers to make use of hardened encryption and identity keys in endpoint devices through a simple API. To support this system we manage the registration of identity keys and a set of device management services for attestation, backup and device grouping.

Rivetz consists of:

- A client module that exposes a handful of privacy, identity and authorization functions implemented in device hardware.
- A web service hosted at Rivetz.net that enables enrolment and pairing of devices and services
- A protocol by which instructions are communicated to a device from a service provider

Rivetz.net will further provide services built on this framework for device management, backup, attestation, etc.

Rivetz.net is a JSON API written in Python that uses the Rivetz private key to enrol the identity keys of devices and services providers. During enrolment the public key of the device or service provider is recorded by Rivetz. Enrolment enables Rivetz to pair a device with a service provider. The result of pairing is that a device has a service public key, endorsed by Rivetz and can therefore respond to service provider instructions.

The Rivetz protocol specifies the structure of an instruction and the signing/encryption that must be applied for the device to accept it. The instruction itself is prepared as a C structure that contains the instruction code, version data and payload. The entire structure is signed by the service provider key and delivered to the Rivet by calling a device local command

Rivetz uses a secure socket to maintain a persistent connection with all riveted devices. This channel is used for pairing and other administrative functions.

Rivetz provides library code to service providers for simplifying the construction and signing of an instruction. This library will be initially provided in Python. Other languages will follow.

3. Tenets

- We provide tools to the web community - Our customers are the vast number of web services and apps that need reliable device authentication and real crypto. In large part this community understands "sign" and "encrypt" and gets lost when asked to specify how. We will decide for them.

- We cannot be a point of failure - Rivetz cannot be another system to which you transfer your trust. We play a valued role in enrolment, pairing and management services (and the Rivet itself), but our server should not be depended on for every transaction.

- We do not track users - Our system is designed to manage devices. We do not identify or track the users that operate them.

- We only trust hardware - Rivetz only puts trust in cryptographic primitives backed by hardware. When not available we will not attempt to "harden" a weak root, but rather will be upfront about the trust level of the endpoint.

4. System Components

This documentation is divided into the discreet components that comprise our system. For each component we describe the functions it exposes, the data it manages and the implementation decisions behind its actualization.

It is the intent of Rivetz to maintain no mission critical data, but rather to provide a platform for seamless yet very secure connections between service providers and devices. On one end is the RivetzEncoder which prepares an instruction for a device, and at the other is the DeviceRivet which is the TEE applet that can act on that instruction. The RivetzProtocol defines how these instructions and replies are constructed

Title Of New Component: Submit

Component	Definition
Device Rivet	The Rivetz TEE applet that embodies our binding between the physical and digital works. The Device Rivet locks features of identity, transaction and attestation to hardware and forms the basis of our technical offering.
Ring Manager	The Ring Manager is a service provided to end-users for managing collections (or Rings) of devices. Devices may be grouped into a single identity and used to backup and endorse each other. Rings may be associated with other rings to create a network of devices.
Rivet Adapter	The RivetAdaptor is the interface between the DeviceRivet bolted into the TEE and the outside world of partner apps and online services. In implementation it manifests in one or more diverse forms. While we strive to present the same basic capabilities across devices, hardware support and OS architecture will dictate what's actually possible and how these features are presented.
Rivetz Encoder	The RivetzEncoder produces a InstructionRecord and processes a ResponseRecord. These are message data structures that are defined to, and interpreted by, the DeviceRivet (the trustlet).
Rivetz Net	The RivetzNet is a service operated by Rivetz for pairing devices and service providers into an endorsed relationship.

5. System Functions

Please refer to the RivetzUseCases

6. Ring Manager

The Ring Manager is a service provided to end-users for managing collections (or Rings) of devices. Devices may be grouped into a single identity and used to backup and endorse each other. Rings may be associated with other rings to create a network of devices.

- Ring Manager
  - Component Context
  - Component Diagram
  - Component Decomposition
  - Entity Responsibility
  - Interface Specification

7. Component Context

(package, patterns, frameworks, preconditions, usage)

8. Component Diagram

9. Component Decomposition

Title Of New Component: ▭ Submit

Component definition

10. Entity Responsibility

(the business or technical entities controlled by this component)

11. Interface Specification

12. Rivetz Net

The RivetzNet is a service operated by Rivetz for pairing devices and service providers into an endorsed relationship.

Originally we intended to put device registration into Namecoin for permanence and transparency, but privacy concerns shelved this plan for the time being. As we begin to collect attestation data on devices this decision will be reassessed. (see the topic history for detail).

- Rivetz Net
    - Component Context
        - Web API
        - Private Key
    - Entity Responsibility
    - Interface Specification
        - Register Device
        - Register Service Provider
        - Get Device ID
        - Pair Device
    - Use Case Reference

13. Component Context

RivetzNet is the first service provider registered to a device and has the special capability of being able to pair additional service providers with that device.

14. Web API

All communications with the Web API need to be authenticated. We could use an API key or better yet, an SSL key swap. We could ask that all requests be signed, but we have to be cognizant of keeping our system simple to use.

15. Private Key

Rivetz relationship with devices is dependent on being able to sign instructions with our private. It is of course paramount that we protect this key. We should seek to encase the key in an HSM.

16. Entity Responsibility

(the business or technical entities controlled by this component)

Title Of New Entity: □ Submit

Entity	Definition
Device ID	The unique identifier, in a UUID, assigned to a device by the RivetzNet or other RegistrationAgent.
Device Pointer	An ephemeral pointer to a device that can be requested by any local application. The DevicePointer can identify a current socket session to RivetzNet and therefore can be used to establish a device communication channel and to look up the permanent identifier, the DeviceID.
Device Registration Record	The root of a device registration includes a unique, anonymous identifier, a registration date, a public key paired to a private key held in the device hardware and an endorsement signature from the RegistrationAgent (assumed to be Rivetz for now.)
DispatchID	The unique identifier used to match an InstructionRecord sent from RivetzNet to a ResponseRecord returned by the RivetAdaptor
Rivetz Coin Account	The RivetzNet uses a block chain infrastructure (currently Namecoin) to store, stamp and publish its registrations. This works by purchasing a name/value pair record in the block chain and thus must have an originating account. The fact that a Rivetz controlled account purchased a record is interpreted as endorsement.

Rivetz Identity Key	Unique public/private key pair generated to represent the endorsement of Rivetz Corp. This key pair should be rotated often and protected in hardware. Ideally, our protocol's would be such that even if the key pair is stolen, the system is not unduly compromised.
Service Provider ID	The unique identifier assigned to a ServiceProvider by RivetzNet.
Service Provider Registration Record	A record created for each registered Service Provider that wants to send instructions to a Riveted device. This includes the service provider name, registration date, public key and endorsement signature (by Rivetz).

17. Interface Specification

18. Register Device

Given unique identifier and a public key, purchase a record of this binding in the block chain. The purchase is made with RivetzCoinAccount thus endorsing the registration. Ideally, the Rivetz signature would only be applied if the device can supply an endorsement key from the OEM.

19. Register Service Provider

Creates a service provider ID for the given organization. The registration must also include the URL where the SP hosts its implementation of RivetzEncoder and the public identity to verify communications.

20. Get Device ID

Given a DevicePointer returns the DeviceID known to the requesting ServiceProvider.

Arguments	definition
Service Provider ID	The unique identifier assigned to a ServiceProvider by RivetzNet.
Device Pointer	An ephemeral pointer to a device that can be requested by any local application. The DevicePointer can identify a current socket session to RivetzNet and therefore can be used to establish a device communication channel and to look up the permanent identifier, the DeviceID.

Returns: DeviceID

21. Pair Device

Before a ServiceProvider can send an instruction it must register its id and public key with the target device. This enables the device to confirm the origin of an instruction before executing it. Pairing a device will automatically create a new identity key on the device

Arguments	definition
Service Provider ID	The unique identifier assigned to a ServiceProvider by RivetzNet.
Device Pointer	An ephemeral pointer to a device that can be requested by any local application. The DevicePointer can identify a current socket session to RivetzNet and therefore can be used to establish a device communication channel and to look up the permanent identifier, the DeviceID.

22. Use Case Reference

- RegisterDeviceWithRivetz - Before a Rivet can do anything it needs to register with RivetzNet. Registration results in the generation of a unique identity key. Registration relies on an endorsement...
- RegisterDeviceWithServiceProvider - A service provider needs to have their ServiceProviderID and public identity key registered with a device before that device will respond to any requests. Even in...
- RegisterServiceProviderWithRivetz - Anyone seeking to code to the Rivetz system needs to register as a ServiceProvider Initial registration is a simple as filling out a form on RivetzNet (http://rivetz...

WebHome > AcronymTable > HSM

A Hardware Security Module is a physical computing device that safeguards and manages digital keys for strong authentication and provides cryptoprocessing.

1. Device ID

The unique identifier, in a UUID, assigned to a device by the RivetzNet or other RegistrationAgent.

2. Device Pointer

An ephemeral pointer to a device that can be requested by any local application. The DevicePointer can identify a current socket session to RivetzNet and therefore can be used to establish a device communication channel and to look up the permanent identifier, the DeviceID.

Datatype:

3. Rivetz Identity Key

Unique public/private key pair generated to represent the endorsement of Rivetz Corp. This key pair should be rotated often and protected in hardware. Ideally, our protocol's would be such that even if the key pair is stolen, the system is not unduly compromised.

4. Device Registration Record

The root of a device registration includes a unique, anonymous identifier, a registration date, a public key paired to a private key held in the device hardware and an endorsement signature from the RegistrationAgent (assumed to be Rivetz for now.)

5. Dispatch ID

The unique identifier used to match an InstructionRecord sent from RivetzNet to a ResponseRecord returned by the RivetAdaptor

6. Rivetz Coin Account

The RivetzNet uses a block chain infrastructure (currently Namecoin) to store, stamp and publish its registrations. This works by purchasing a name/value pair record in the block chain and thus must have an originating account. The fact that a Rivetz controlled account purchased a record is interpreted as endorsement.

7. Service Provider ID

The unique identifier assigned to a ServiceProvider by RivetzNet.

8. Service Provider Registration Record

A record created for each registered Service Provider that wants to send instructions to a Riveted device. This includes the service provider name, registration date, public key and endorsement signature (by Rivetz).

9. Rivetz Encoder

The RivetzEncoder produces an InstructionRecord and processes a ResponseRecord. These are message data structures that are defined to, and interpreted by, the DeviceRivet (the trustlet).

a. Component Context

The RivetzEncoder is software written to be hosted by our partners.

The RivetzEncoder is distributed as public open source.

b. Entity Responsibility

Title Of New Entity: ☐ Submit

c. Interface Specification

d. Implementation

e. Use Case Reference

EncryptSomething - Rivetz provides the mechanics for encrypting text or images but expects partners to project the interface for their service, whether it be a messaging application.

10. Service Provider Identity Key

The private portion of the service provider identity is used by the RivetzEncoder to sign instructions. The public portion is provided to Rivetz and to paired devices.

11. Device Rivet

The Rivetz TEE applet that embodies our binding between the physical and digital works. The Device Rivet locks features of identity, transaction and attestation to hardware and forms the basis of our technical offering.

- Device Rivet
  - Component Context

- Component Description
- Entity Responsibility
- Interface Specification
  - Enroll Device
  - Generate Key
  - Encrypt with Key
  - Decrypt with Key
  - Process Instruction
- Use Case Reference
- Notes

a. Component Context

We currently have two target platforms for hosting the DeviceRivet implementation: Trustonic on Android and Intel ME for Windows PC's. Both environments have limited processing and are specifically architected to be simple for the sake of security and resource usage.

Trustonic Trusted Apps (TA's) are implemented with the Android NDK compiler in C. Interfacing with the TA is done using a shared memory buffer. Commands are packed into a memory block and notification is sent to the Trustonic controller to load and execute the TA. Notification is synchronous. The host app (a regular Android app) waits for response. A Trusted App is expected to store its data on the host, however, the Trustonic controller provides a secure wrapper so that the data can only be opened when running in the TEE.

For the Intel implementation apps are written in Java and signed by Intel's master key. We were able to get the DAL SDK from Intel for this purpose and they began in December to show active support from our efforts.

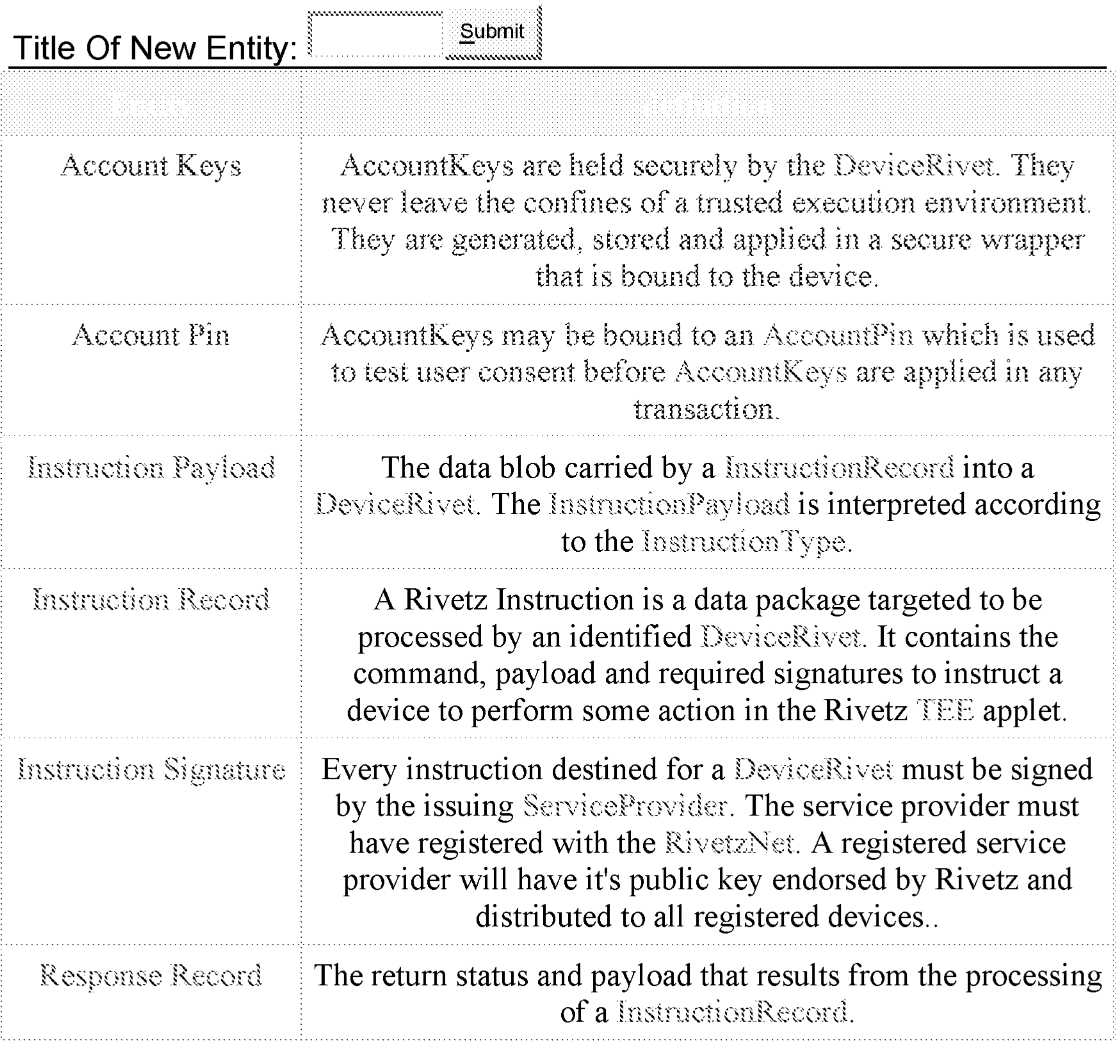
b. Component Description

The implementation is quite different across platforms and integration with the RivetAdaptor will further incur device specific methods. However, the logical implementation is intended to be the same and the input data structures are by necessity the same. The rest of the Rivetz system would like to treat devices as all supporting the same interface, yet some with more or less feature sets.

There are three main areas of functionality in the DeviceRivet (the Trustlet):

- Device Enrollment - This is the process by which the DeviceRivet establishes an identity with the RegistrationAgent (the RivetzNet).
- Instruction Processing - Execute a given instruction. This is a signed data structure that originates from a ServiceProvider.
- Security Primitives - Simple security functionality exposed for local application usage.

c. Entity Responsibility



Title Of New Entity:

Entity	Definition
Account Keys	AccountKeys are held securely by the DeviceRivet. They never leave the confines of a trusted execution environment. They are generated, stored and applied in a secure wrapper that is bound to the device.
Account Pin	AccountKeys may be bound to an AccountPin which is used to test user consent before AccountKeys are applied in any transaction.
Instruction Payload	The data blob carried by a InstructionRecord into a DeviceRivet. The InstructionPayload is interpreted according to the InstructionType.
Instruction Record	A Rivetz Instruction is a data package targeted to be processed by an identified DeviceRivet. It contains the command, payload and required signatures to instruct a device to perform some action in the Rivetz TEE applet.
Instruction Signature	Every instruction destined for a DeviceRivet must be signed by the issuing ServiceProvider. The service provider must have registered with the RivetzNet. A registered service provider will have it's public key endorsed by Rivetz and distributed to all registered devices..
Response Record	The return status and payload that results from the processing of a InstructionRecord.

d. Interface Specification

i. Enroll Device

ii. Generate Key

iii. Encrypt with Key

The TEEAdapter looks up named encrypt key in the ServiceProviderRecord

iv. Decrypt with Key

v. Process Instruction

e. Use Case Reference

- CreateKey - Create a key pair in the DeviceRivet for either signing and encrypting. Actors ServiceProvider Description The primary purpose of Rivetz is to secure and apply...
- CreateLocalUser - Establish a local entity that can authorize use of the Rivet in cases where no ServiceProvider authorization is given Actors Select/create Actors from ProductActors...
- EncryptSomething - Rivetz provides the mechanics for encrypting text or images but expects partners to project the interface for their service, whether it be a messaging application...
- RegisterDeviceWithRivetz - Before a Rivet can do anything it needs to register with RivetzNet. Registration results in the generation of a unique identity key. Registration relies on an endorsement...

12. Instruction Payload

The data blob carried by an InstructionRecord into a DeviceRivet. The InstructionPayload is interpreted according to the InstructionType.

13. Instruction Record

A Rivetz Instruction is a data package targeted to be processed by an identified DeviceRivet. It contains the command, payload and required signatures to instruct a device to perform some action in the Rivetz TEE applet.

Most instructions will result in the construction and return of a ResponseRecord. This will be delivered back to the ServiceProvider by the RivetzDispatch.

a. Data Structure

Parameter	Type/Size	Description
VersionID	integer	The version id type for data structure for compatibility
ServiceProviderID	UUID	The unique identifier of the service provider issuing this instruction
InstructionType	integer	The instruction type identifier. This determines how to interpret the contents of the payload
InstructionPayload	blob	Arbitrary data blob
InstructionSignature	byte(512)	Hash of the instruction signed by the service provider key

b. Instruction Types

Type Name	Value	Description
RIVETZ_DO_TEXT_CONFIRMATION		The payload contains a text message and a signed hash. The message string will be displayed along with a confirmation and cancel button. On confirmation, the device will sign the message and return it.
RIVETZ_DO_IMAGE_CONFIRMATION		The payload contains an image and a signed hash. The image will be displayed along with confirm and cancel buttons.
RIVETZ_DISPLAY_IMAGE		The payload contains an image encrypted with the device key and a hash signed by the publisher. The image is displayed by the DeviceRivet. There is no return.
RIVETZ_DISPLAY_TEXT		The payload contains text encrypted with the device key and a hash signed by the publisher. The text is rendered by the DeviceRivet. There is no return.

RIVETZ_CREATE_BITCOIN_ACCOUNT		A new bitcoin account is created and the public address is returned
RIVETZ_UPDATE_SP_LIST		The payload contains the ID's and public keys of the service providers that have been registered by the RegistrationAgent (That's Rivetz). This list is signed by the RegistrationAgent that registered the device. In other words, only the system who registered the device can update the list of registered service providers.
RIVETZ_SIGN_VC_TXN	0x0001	The payload contains a fully populated Virtual Coin (Bitcoin, Litecoin, Peercoin, etc) transaction that is to be signed with the named Bitcoin account key maintained by the DeviceRivet.
RIVETZ_ADD_KEY	0x0101	The payload contains data to add an existing key to the Service Provider Key List. Recommended you create a new key so that it is never seen in the normal world.
RIVETZ_GET_KEY	0x0102	The payload contains the request to retrieve the public key from a KeyRecord.
RIVETZ_DELETE_KEY	0x0103	The payload contains the request to delete a KeyRecord.
RIVETZ_ENUM_KEY	0x0104	The payload contains the request to get a list of KeyRecords.
RIVETZ_ECDSA_CREATE	0x0201	The payload contains the request to create a ECDSA public and private Keys. Key is stored in KeyRecord in RivetAndroid.

RIVETZ_ECDSA_SIGN	0x0202	The payload contains the request to sign data using a ECDSA private key.
RIVETZ_ECDSA_VERIFY	0x0203	The payload contains the request to verify data using a ECDSA public key.
RIVETZ_ECDSA_GETPUBPRV	0x0204	The payload contains the request to get the public virtual coin (Bitcoin, Litecoin, Peercoin, etc) address from a ECDSA private key.
RIVETZ_ECDSA_GETPUBSIG	0x0205	The payload contains the request to get the ECDSA public key from a signature and message.
RIVETZ_ECDH_ENCRYPT	0x0301	The payload contains the request to encrypt data using ECDH.
RIVETZ_ECDH_DECRYPT	0x0302	The payload contains the request to decrypt data using ECDH.

Note that not all devices will be able to support all instructions. If the instruction is not supported the DeviceRivet will return NOT_SUPPORTED. See ResponseRecord.

14. Instruction Type

A constant value that indicates the type of the InstructionRecord. This determines how the InstructionPayload is to be interpreted.

Instruction Types are described in InstructionRecord.

15. Instruction Signature

Every instruction destined for a DeviceRivet must be signed by the issuing ServiceProvider. The service provider must have registered with the RivetzNet. A registered service provider will have its public key endorsed by Rivetz and distributed to all registered devices.

16. Account Keys

AccountKeys are held securely by the DeviceRivet. They never leave the confines of a trusted execution environment. They are generated, stored and applied in a secure wrapper that is bound to the device.

17. Account Pin

AccountKeys may be bound to an AccountPin which is used to test user consent before AccountKeys are applied in any transaction.

18. Response Record

The return status and payload that results from the processing of an InstructionRecord.

a. Status Codes

Return Code Name	Description
RETURN_INSTRUCTION_EXECUTED	A generic return for an instruction that was executed by the DeviceRivet.
RETURN_NOT_SUPPORTED	The InstructionType provided in the InstructionRecord is unsupported on this device
RETURN_NOT_KNOWN	The InstructionType provided in the InstructionRecord is unknown

RETURN_CONFIRMATION_OK	The request for confirmation was confirmed by the user. The payload of the return will include a hash of the confirmation object (image or text) signed by the device
RETURN_CONFIRMATION_CANCELLED	The request for confirmation was cancelled by the user
RETURN_CONFIRMATION_EXPIRED	The request for confirmation was neither confirmed nor cancelled by the user within the time limit

19. Rivet Adapter

The RivetAdaptor is the interface between the DeviceRivet bolted into the TEE and the outside world of partner apps and online services. In implementation it manifests in one or more diverse forms. While we strive to present the same basic capabilities across devices, hardware support and OS architecture will dictate what's actually possible and how these features are presented.

- Rivet Adapter
  - Diagram
  - Sub-Components
  - Implementation
  - Use Case Reference

a. Diagram

b. Sub-Components

The RivetAdaptor is composed of outward and inward looking interfaces. The inward looking interface, the TEEAdapter, handles proprietary communications with the trustlet (the DeviceRivet). The HostAdaptor is provided to expose services to third-party applications.

Please refer to the individual sub-components for interface and implementation details.

Host Adapter **-- The HostAdaptor presents the interface of the RivetAdaptor through different local contexts, such as browsers or system services. Multiple realizations for diverse contexts are anticipated though initially this is an Android service and a windows com process.**

Socket Adapter **-- Connects the client environment to RivetzNet.**

TEE Adaptor **-- This component is the proprietary glue that pipes commands into our trustlet running in Trustonic or Intel ME.**

c. Implementation

In the Android implementation the RivetAdaptor manifests as an Android NDK service app. It is configured to launch at boot. The RivetAdaptor prepares message buffers that are piped to the Trustlet and then synchronously awaits notification of a response event. The manifest of the Android app presents a series of intents for a third-party to trigger. The app, the NDK binaries and the Trustlet are all packaged into a single APK for distribution.

d. Use Case Reference

- CreateLocalUser - Establish a local entity that can authorize use of the Rivet in cases where no ServiceProvider authorization is given Actors Select/create Actors from ProductActors...
- EncryptSomething - Rivetz provides the mechanics for encrypting text or images but expects partners to project the interface for their service, whether it be a messaging application...
- RegisterDeviceWithRivetz - Before a Rivet can do anything it needs to register with RivetzNet. Registration results in the generation of a unique identity key. Registration relies on an endorsement...
- RegisterDeviceWithServiceProvider - A service provider needs to have their ServiceProviderID and public identity key registered with a device before that device will respond to any requests. Even in...

20. Host Adapter

The HostAdaptor presents the interface of the RivetAdaptor through different local contexts, such as browsers or system services. Multiple realizations for diverse contexts are anticipated though initially this is an Android service and a windows com process.

The HostAdaptor is primarily there to isolates the TEEAdapter from the host environment. However it does have a minimal UI presence on the host machine. It presents the "About" page and is the item the end user can identify in their apps list.

Eventually the HostAdaptor will present RingManager services such as *backup* or *join*.

- Host Adapter
    - Interface
        - GetPointer
        - GetHash
        - Execute
        - Encrypt
        - Decrypt
    - Android Implementation
        - Android Intent Documentation
    - Windows Implementation
    - Use Case Reference

a. Interface

The HostAdaptor operates in a potentially hostile environment. We will therefore typically have limited assurance that the client has not been compromised. The HostAdaptor's role is therefore primarily to facilitate easy access to the DeviceRivet. Instructions from a ServiceProvider intended for the DeviceRivet will be signed by the ServiceProvider and then passed through to the TEEAdapter and DeviceRivet through an *Execute* instruction. Instructions intended to use the LocalServiceProvider role may be constructed by the HostAdaptor and then signed by the TEEAdapter or other entity prior to the instruction being passed to the DeviceRivet.

Certain local services such as Encrypt and Decrypt are allowed to be called using the LocalServiceProvider role and the HostAdaptor provides an interface for these services locally for the convenience of our customers. These may be disallowed on certain platforms.

i. GetPointer

We want to protect the permanent device identifiers from abuse. A validated service provider will need to ask, "what device is this?" So that a rogue app cannot get a useful response with the same question we use a DevicePointer. The DevicePointer is an identifier that's only valid during a socket connection with RivetzNet. With the DevicePointer in hand, the ServiceProvider can query RivetzNet directly for the permanent DeviceID or to request pairing. The SocketAdaptor stores the DevicePointer in memory whenever it connects to RivetzNet.

Argument
none

Return: Device Pointer -- An ephemeral pointer to a device that can be requested by any local application. The DevicePointer can identify a current socket session to RivetzNet and therefore can be used to establish a device communication channel and to look up the permanent identifier, the DeviceID.

ii. GetHash

For signing and encrypting instructions the ServiceProvider needs to sign a hash of the object.

Argument	Definition
Data Blob	Data as an unspecified collection of bytes of any length

Return: **SignedHash –**

iii. Execute

Passes an InstructionRecord to the TEEAdapter and returns ResponseRecord. The Rivet will need the given the context in which to process the instruction so it needs the ServiceProviderID passed in the clear.

Argument	Definition
Service Provider ID	The unique identifier assigned to a ServiceProvider by RivetzNet.
Instruction Record	A Rivetz Instruction is a data package targeted to be processed by an identified DeviceRivet. It contains the command, payload and required signatures to instruct a device to perform some action in the Rivetz TEE applet.

Return: Response Record **-- The return status and payload that results from the processing of an InstructionRecord.**

iv. Encrypt

Argument	Definition
Service Provider ID	The unique identifier assigned to a ServiceProvider by RivetzNet.
EcryptPublicKey	
Data Blob	Data as an unspecified collection of bytes of any length

Return: Data Blob -- Data as an unspecified collection of bytes of any length

v. Decrypt

Argument	Definition
Service Provider ID	The unique identifier assigned to a ServiceProvider by RivetzNet.
Data Blob	Data as an unspecified collection of bytes of any length

Return: Data Blob -- Data as an unspecified collection of bytes of any length

b. Android Implementation

The HostAdaptor is the standard Java portion of the Rivetz client for Android. It exposes it's interface through *Intents*, the standard mechanism for cross app communications. For example:

```
public void connectRivet(String serviceProviderID, ByteArray
instruction) {
    Intent intent = new Intent(com.rivetz.RivetActionExecute)
            .putExtra(com.rivet.RivetAction.EXTRA_SPID,
serviceProviderID)
            .putExtra(com.rivet.RivetAction.EXTRA_INSTRUCTION,
instruction);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

Each action is defined as a separate class that inherits from `com.rivetz.RivetAction`. For example:

```
public class RivetActionInstruction extends RivetAction { //
RivetAction extends Activity
        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);

                // Get the intent that started this activity
                Intent intent = getIntent();
                int SpID =
intent.getStringExtra(com.rivet.RivetAction.EXTRA_SPID,0);
                ByteArray instruction =
intent.getStringExtra(com.rivet.RivetAction.EXTRA_INSTRUCTION,0);
                // call the corresponding JNI function
                result =
Trustlet.RivetzActionPair(SpID,instruction);

          }
```

The TEEAdapter defines the JNI (Java Native Interface) code that passes an instruction through to the DeviceRivet.

i. Android Intent Documentation

These definitions are pulled into the SDK pages for public display. See RivetzAndroidClient.

Title Of New Android Intent: ▭ Submit

AndroidIntent	definition
INSTRUCT_CREATEKEY	Create a key of the specified type. Rivetz stores the key in a local hardware encrypted storage space unique to the Service Provider. Keys are named for future reference.
INSTRUCT_DECRYPT	Decrypts the given data object with the named key
INSTRUCT_DELETEKEY	Removes the key identified by KeyName from the Service Provider's key sets
INSTRUCT_ENCRYPT	Encrypts the given data object with the named key. Generally this is used with a public key loaded through INSTRUCT_LOADKEY.
INSTRUCT_EXECUTE	Provide a server-signed instruction to a device. While the Rivet may be tasked with local unsigned requests, ideally instructions are signed by a service provider key established during service provider registration.
INSTRUCT_GETKEY	Gets the key data from the named key stored in the Rivet. Results will vary based on the KeyType. Symmetric keys and private keys are returned encrypted with a unique key protected by the device hardware.

INSTRUCT_GETPOINTER	Get a termporary unique pointer to the device that can be used for making web requests to Rivetz.Net
INSTRUCT_GETPUBPRV	Summary
INSTRUCT_GETPUBSIG	Summary
INSTRUCT_KEYENUM	Summary
INSTRUCT_LOADKEY	Loads an arbitrary public key into the Service Provider key set for use with INSTRUCT_ENCRYPT
INSTRUCT_REGISTERPROVIDER	A Service Provider needs to register or pair with a device before the Rivet will respond to any instructions. This process is essentially a key exchange ceremony brokered by Rivetz.net.
INSTRUCT_SIGN	Sign a blob of data with the named key. The algorithm to be used is established when the key is created.
INSTRUCT_SIGNTXN	Sign a coin transaction with the named coin (wallet) key
INSTRUCT_VERIFY	Verify a signature for a given object. Result code: Rivet.RESULT_OK signifies that the signature passed.

c. Windows Implementation

TBD

d. Use Case Reference

CreateLocalUser - Establish a local entity that can authorize use of the Rivet in cases where no ServiceProvider authorization is given

Actors Select/create Actors from ProductActors...

- EncryptSomething - Rivetz provides the mechanics for encrypting text or images but expects partners to project the interface for their service, whether it be a messaging application...

-

21. Socket Adapter

Connects the client environment to RivetzNet.

- Socket Adapter

a. Component Context

b. Entity Responsibility

Title Of New Entity: ☐ Submit

Entity	Definition
Rivetz Net URL	The URL where we host RivetzNet
Session Object	Defines the keys and other data for a temporal session between two secure endpoints.

c. Interface Specification

i. Connect

Open a connection with the server. The server will return a DevicePointer assigned to this session. Connect is called when the RivetAdaptor starts.

Arguments: none

Returns: none

ii. Disconnect

Disconnect from the server and discard the DevicePointer.

Arguments: none

Returns: none

iii. GetPointer

Return the current DevicePointer or null if there is no session.

Arguments: none

Returns: Device Pointer -- An ephemeral pointer to a device that can be requested by any local application. The DevicePointer can identify a current socket session to RivetzNet and therefore can be used to establish a device communication channel and to look up the permanent identifier, the DeviceID.

iv. Instruct

Receive an InstructionRecord from RivetzNet, pass it to the rivet and asynchronously post the ResponseRecord. Every instruction will come with a unique DispatchID that is used by RivetzNet to match the Instruction to the response. Note that some instructions may involve user interaction through the TUI and therefore may incur considerable elapsed time before a response is posted.

Argument	Definition
DispatchID	The unique identifier used to match an InstructionRecord sent from RivetzNet to a ResponseRecord returned by the RivetAdaptor
Service Provider ID	The unique identifier assigned to a ServiceProvider by RivetzNet.
Instruction Record	A Rivetz Instruction is a data package targeted to be processed by an identified DeviceRivet. It contains the command, payload and required signatures to instruct a device to perform some action in the Rivetz TEE applet.
Returns	Definition
DispatchID	The unique identifier used to match an InstructionRecord sent from RivetzNet to a ResponseRecord returned by the RivetAdaptor
Response Record	The return status and payload that results from the processing of a InstructionRecord.

d. Use Case Reference

22. TEE Adaptor

This component is the proprietary glue that pipes commands into our trustlet running in Trustonic or Intel ME.

a. Design Concepts

The Trustonic and Intel ME environments follow the same basic architecture: the host system serializes data into a memory buffer and then triggers the TEE to process. This is a blocking (synchronous) request. Control is returned when the TEE exits, presumably after writing response data in the memory buffer.

As our TEE code can do more than one thing, part of the data structure passed in needs to identify the procedure to execute. This in turn determines how the rest of the data structure is interpreted.

Likewise, the instruction being executed needs context data that provides the keys to work with. As the TEE has no native persistent memory, data records are encrypted by the TEE and given to the TEEAdapter to store and return when needed. Records are stored per ServiceProvider and include the device identity, wallet and encryption keys unique to the given service provider

b. Component Diagram

All the work happens in the TEE Loader where data from parameters and storage is serialized into a structure to be passed via shared memory to the TEE environment.

i. TEE Communication Record

For every request, the TEE Adapter takes the input, packages a data structure for the TEE and calls execute on the Trusted Applet environment. When the execution is complete, the shared memory is recast as a response record. Any return data is prepared for the original calling function and the ServiceProvider Record is stored back to disk.

c. Entity Responsibility

Title Of New Entity: ☐ Submit

Data	definition
Service Provider Record	The Service Provider context information that is provided to the TEE when it processes an instruction.

d. Interface Specification

i. Process Instruction

Called by the SocketAdaptor when it receives an instruction from the RivetzEncoder. The instruction is a packaged blob meant to be processed directly by the TEE without parsing.

argument	definition
Service Provider ID	The unique identifier assigned to a ServiceProvider by RivetzNet.
Instruction Record	A Rivetz Instruction is a data package targeted to be processed by an identified DeviceRivet. It contains the command, payload and required signatures to instruct a device to perform some action in the Rivetz TEE applet.

The TeeAdaptor will load the ServiceProviderRecord, serialize it into a memory buffer along with the InstructionRecord and the trigger the TEE to process. On TEE exit, the ServiceProviderRecord is written back to disk and the response blob is returned to the SocketAdaptor.

ii. Encrypt

Local request to encrypt using a named key. Encryption keys belong to a ServiceProviderRecord and are created using the CreateKey instruction.

argument	definition
Service Provider ID	The unique identifier assigned to a ServiceProvider by RivetzNet.
Key Name	An arbitrary string assigned to a key created in the Rivet.
Data Blob	Data as an unspecified collection of bytes of any length

iii. Decrypt

Local request to decrypt using a named key.

argument	definition
Service Provider ID	The unique identifier assigned to a ServiceProvider by RivetzNet.
Key Name	An arbitrary string assigned to a key created in the Rivet.
Data Blob	Data as an unspecified collection of bytes of any length

e. Android Implementation

The Android implementation uses the Java Native Interface (JNI) implemented by the Android NDK.

In order to communicate with the Trustonic applet, the DeviceRivet, we need to use Android JNI code. Each intent fired on a RivetAction will have a corresponding JNI function defined that takes us into a C++ implementation environment.

```
EXTERN_C JNIEXPORT jstring JNICALL

Java_com_rivetz_Trustlet_RivetzActionPair(JNIEnv *env, jobject obj,
jstring messageIn) {

        /* implementation */

}
```

f. Use Case Reference

23. Service Provider Record

The Service Provider context information that is provided to the TEE when it processes an instruction.

a. Structure

This topic is just for getting the concepts down.

Attribue	definition

Service Provider ID	The unique identifier assigned to a ServiceProvider by RivetzNet.
Key Record	A persistent object that stores the TEE keys in the RivetAdaptor environment. Each key is created on behalf a ServiceProvider and given a name and a usage rules.

b. Realization

This is expected to be a flat file of binary data that can be easily serialized into and back out of the TEE memory buffer.

Details and datatypes are defined and maintained in the source code at GitHub. See https://github.com/rivetz/RivetzEncoder/blob/master/riv_types.h

24. Rivetz Protocols

DeviceEnrollmentProtocol

InstructionProcessingProtocol

IntercedeOnboardingProcess

25. Instruction Processing Protocol

a. Overview

The counterpart to the DeviceRivet is the RivetzEncoder. The RivetzEncoder prepares a command to be executed by a specific device which is signed and/or encrypted by the ServiceProvider. The ServiceProvider public keys are preloaded into the device during a pairing process conducted by RivetzNet. This allows the DeviceRivet to validate the origin of the request, and if needed decrypt the contents of the instruction.

The sequence of packaging and delivering an instruction is pretty straightforward. The ServiceProvider generates an InstructionRecord with the help of RivetzEncoder libraries. The instruction includes the type, the target device and payload. The instruction may be encoded with the device key and must be signed by the service provider key. The device key is fetched from RivetzNet, or directly from the block chain, by looking up the DeviceRegistrationRecord.

26. Device Enrollment Protocol

a. Overview

Device enrollment is the pedestal on which our entire ecosystem stands.

27. Intercede Onboarding Process

The following roughly describes the steps that Rivetz will need to complete to start using Intercede for installing a DeviceRivet.

See IntercedeGroup for background and docs.

- Intercede Onboarding Process
  - KEY SETUP:
  - BUILD DEVICE RIVET APPLICATION
  - Execution
    - Transport Key
    - Personalization Master Key
    - Key Verification
    - Purchase Receipt Key

a. KEY SETUP:

- First create a test Transport Key (we'll call this the TTK).
- Generate three random 256-bit values and store them as Share1, Share2, Share3
  - Perform an XOR operation between the shares (Share1 XOR Share2 XOR Share3) to obtain the TTK.
- Create files for each of the three shares and encrypt them individually with the three PGP keys that Intercede sent to Rivetz.

- Generate a 256-bit test Personalization Master Key (a TPMK) and store this in Rivetz code somewhere.

- Encrypt the TPMK with the TTK as described in the Intercede document and send this via e-mail to Intercede.

- Generate a test Purchase Receipt Key (TPRK).

- Generate a "customer reference" number for Rosie Wallet or whatever test Service Provider we want.

- Send the public portion of the TPRK (we can call this the TPRPK) to Intercede.

b. BUILD DEVICE RIVET APPLICATION

- We should modify the current DeviceRivet software to be able to accept a personalization package. The personalization package will contain a key that is derived from the TPMK.

- Create software on the Rivetz.net server side that derives the personalization key for each individual DeviceRivet.

- Update the Rivetz provisioning protocols to use the shared DeviceRivet personalization key to establish trust between the device and Rivetz.net. This will likely involve the DeviceRivet generating new device-specific keys and signing/encrypting those for Rivetz.net with the personalization key for that particular DeviceRivet.

- Include the MyTAM client library in our real world application (the RivetAdaptor) to assist in installing the DeviceRivet and personalization package.

c. Execution

i. Transport Key

To build the random values, share1, share2, share2:

```
tr -cd [:alnum:] < /dev/urandom | head -c $(tr -cd 0-9 <
/dev/urandom | head -c 1) | sha256sum | tr -d ' -'
```

It should look like this:

`a9f51566bd6705f7ea6ad54bb9deb449f795582d6529a0e22207b8981233ec58.`
What this command does is pipe the linux kernel random data through a text processing tool (tr) that pulls out alphanumeric characters, truncates the result to a random number of characters (with head) and then pipes this into sha256sum. Finally, it uses tr again to remove the trailing space and hyphen

Do this three times and XOR the results together using a python command line call:

```
python -c 'print "{:x}".format(

int("bb65b75d83d82065b17929affd23e0f26f9e134ff90646e1fd087eb4339b89f
e",16) ^

int("e5568e87e6fd44b373fa92c361f5c5c37ce5f4ddf97cafe1177b3d372091285
4",16) ^

int("a9f51566bd6705f7ea6ad54bb9deb449f795582d6529a0e22207b8981233ec5
8",16))'
```

This results in:

`f7c62cbcd842612128e96e2725089978e4eebfbf655309e2c874fb1b01394df2`
What this does is cast each of the hex strings to int, XOR's them together and then formatting the result back into hex

Note that these files are all ASCII hex representation. To translate into binary do

```
cat share1 | xxd -r -p > share1.bin
```

Putting it all together

```
tr -cd [:alnum:] < /dev/urandom | head -c $(tr -cd 0-9 <
/dev/urandom | head -c 1) | sha256sum | tr -d ' -' > share1

tr -cd [:alnum:] < /dev/urandom | head -c $(tr -cd 0-9 <
/dev/urandom | head -c 1) | sha256sum | tr -d ' -' > share2

tr -cd [:alnum:] < /dev/urandom | head -c $(tr -cd 0-9 <
/dev/urandom | head -c 1) | sha256sum | tr -d ' -' > share3
```

```
python -c 'print "{:x}".format(int(open("share1","r").read(),16) ^
int(open("share2","r").read(),16) ^
int(open("share3","r").read(),16))' > TTK
```

Then for each fragment:

```
gpg --import recipient.asc

cat share1 | xxd -r -p > share1.bin

gpg -o encrypted_share_for_recipient.gpg --encrypt -r <KEY-ID>
share1.bin
```

ii. Personalization Master Key

1. generate random number
2. convert to binary
3. encrypt with Transport Key and then pipe into hex format for delivery to Intercede

```
tr -cd [:alnum:] < /dev/urandom | head -c $(tr -cd 0-9 <
/dev/urandom | head -c 1) | sha256sum | tr -d ' -' > TPMK

cat TPMK | xxd -r -p > TPMK.bin

openssl enc -aes-256-ecb -in TPMK.bin -nopad -K `cat TTK` | xxd -p -
c 256 > TPMK.enc.hex
```

iii. Key Verification

A check value (KCV) may also be calculated and sent to Intercede. The optional check value ensures that the Personalization Master Key is correct once imported into the Intercede HSM – the check value is computed as follows.

- Use the (unencrypted) Personalization Master Key to encrypt one block (16 bytes) of binary zero's. (Use ECB mode, no padding.)

- The first 3 bytes of the output are the check value (KCV). Transmit the KCV to Intercede.

- The process of importing the key into MyTAM at Intercede will verify the KCV (if supplied), and provide additional verification that the key exchange has been performed correctly.

```
echo 00000000000000000000000000000000 | xxd -p -r | openssl enc -
aes-256-ecb -nopad -K `cat TPMK` | xxd -p -c 256 | cut -b -6 >
TPMK.kcv
```

iv. Purchase Receipt Key

This is supposed to mimic the Google Play receipt key for in app purchases. The key is used to sign the device SUID during provisioning. Intercede uses this as a receipt of "purchase".

```
openssl genrsa -out TPRK.pem 2048

openssl rsa -in TPRK.pem -pubout > TPRPK.pem
```

This generates a 2048 bit RSA key in the file TPRK.pem and then extracts the public key into TPRPK.pem which is to be sent to Intercede.

From openssl.org: "PEM form is the default format: it consists of the DER format base64 encoded with additional header and footer lines. On input PKCS#8 format private keys are also accepted."

From Google Play documentation: "The Base64-encoded RSA public key generated by Google Play is in binary encoded, X.509 subjectPublicKeyInfo DER SEQUENCE format. It is the same public key that is used with Google Play licensing."

```
openssl genrsa -out TPRK.pem 2048

openssl rsa -in TPRK.pem -outform der -pubout > TPRPK.der
```

This delivers a binary format key

28. Rivetz Use Cases

Rivetz provides an SDK to partners for accomplishing simple yet critical transactions with a device. This spans authentication to messages to Bitcoin signing. The interface is a systems interface but some services will engage the user for PIN entry, visual confirmation, etc.

a. Use Cases

Title Of New Use Case: [ ] Submit

UseCase	Description
Create Bitcoin Account	Generate a new Wallet Account id in the device hardware
Create Key	Create a key pair in the DeviceRivet for either signing and encrypting.
Create Local User	Establish a local entity that can authorize use of the Rivet in cases where no ServiceProvider authorization is given
Decrypt Something	Given an encrypted object and a key name, decrypt the object either for TUI display or to return to the requester.
Encrypt Something	Rivetz provides the mechanics for encrypting text or images but expects partners to project the interface for their service, whether it be a messaging application or some other.
Register Device with Rivetz	Before a Rivet can do anything it needs to register with RivetzNet. Registration results in the generation of a unique identity key.
Register Device with Service Provider	A service provider needs to have their ServiceProviderID and public identity key registered with a device before that device will respond to any requests.
Register Service Provider with Rivetz	Anyone seeking to code to the Rivetz system needs to register as a ServiceProvider
Send a Secure Confirmation Request	Package a short message that will be delivered to the target endpoint device and displayed to the user with secure display if available. The communicated is signed both ways to ensure the confirmation is valid. The message may be an image or text.
Sign Bitcoin	Given a fully formed bitcoin transaction (where the origin account is owned by the target device hardware), sign the

Transaction	transaction and return it. In most cases this should also involve prompting the user for confirmation with secure display, if available, or at least common display otherwise.
Sign Something	Given a named key and object reference, return a signed hash of the object
User Recovers Forgotten Device PIN	Summary
Verify Something	Verify the signature on an object with a named or given key.

b. Actors

Title Of New Actor: [ ] Submit

Actor	Definition
Account Representative	A Rivetz employee responsible for the relationship with a ServiceProvider
Service Provider	Service Provider's use the capabilities provided to Rivetz to enhance their own services. They are our partners and the primary source of income.
Service User	An ServiceUser is someone who is engaged with a primary feature/function of our service.
System Administrator	A System Administrator engages with the installation, configuration and maintenance of our service
Trusted Application Manager	An entity that can load and endorse a trusted application into a trusted execution environment (TEE)

29. Trusted Application Manager

An entity that can load and endorse a trusted application into a trusted execution environment (TEE)

a. Definition

In the world of Trustonic GieseckeAndDevrient and IntercedeGroup are established as TAM's.

30. Service User

A ServiceUser is someone who is engaged with a primary feature/function of our service.

a. Definition

31. System Administrator

A System Administrator engages with the installation, configuration and maintenance of our service

a. Definition

32. Account Representative

A Rivetz employee responsible for the relationship with a ServiceProvider

a. Definition

33. Service Provider

Service Provider's use the capabilities provided to Rivetz to enhance their own services.

Definition

Service Providers need to be registered with the RivetzNet in order to do business with us, or more specifically, to access our API's and sign instructions targeted to riveted devices.

a. Demo Service Provider

It's clear that we need to have a ServiceProviderID that can be easily handed out to developers for early testing and experimentation. We are doing this already, but with a random UUID that MarkHoblit has embedded. For example:

```
Intent intent = new Intent(Rivet.RIVET_INTENT)

                  .putExtra(Rivet.EXTRA_INSTRUCT,
Rivet.INSTRUCT_CREATEKEY)

                  .putExtra(Rivet.EXTRA_SPID, "98f88054-f98c-440c-
81aa-77fa70a31116-fbca7c00-0603-4c1f-a354-820ae9ec46b9")

                  .putExtra(Rivet.EXTRA_KEYTYPE,
Rivet.KEYTYPE_ECDSA_DEFAULT)

                  .putExtra(Rivet.EXTRA_KEYNAME,"MyKey");
```

It should be noted that a device activated with the demo SPID will incur a royalty to Intercede and Trustonic just like a production Rivet.

34. Register Service Provider with Rivetz

Anyone seeking to code to the Rivetz system needs to register as a ServiceProvider

Initial registration is a simple as filling out a form on RivetzNet (http://rivetz.com/docs/registration.html).

a. Actors

ServiceProvider, AccountRepresentative

b. Description

1. Service Provider creates local public/private keys

2. Service provider goes to HTTP form on rivetz.com (http://rivetz.com/docs/registration) and inputs the following information:
    - Company Name
    - Contact: First Name, Last Name, Position, Email, Phone
    - Company Website
    - Company Address: Street, City, State/Province, Country
3. Service Provider Clicks "I Accept" to terms of service agreement.
4. Service Provider selects a password and confirms it (user name will be the given contact email)
    - we tell them that this can be replaced by device authentication later
5. Service Provider is requested to upload a public key
    - This can be skipped and done later
    - We should also provide more secure ways of obtaining the public key than this upload
6. If key is provided, then a SPID (service provider ID) is generated and emailed to the customer
    - If no key is provided an email confirmation is sent with a pending message and instructions on providing the key.
7. AccountRepresentative will receive notification of anew registration
    - At this point the data can be loaded into SalesForce and Account Rep may choose to follow up personally.

i. Variation: New Service Provider Returns to Provide Key

1. Service Provider logs in with email and password
2. Service Provider notes "pending" state of account
3. Service Provider clicks to fix pending state and is prompted with a entry box for their public key
4. Once the key is posted, a SPID is created and emailed to the Service Provider contact email
5. The account is no longer pending

6. AccountRepresentative is notified of the change in the account.

c. Notes

35. User Recovers Forgotten Device PIN

Summary

a. Actors

Select/create Actors from ProductActors

b. Description

c. Notes

36. Verify Something

Verify the signature on an object with a named or given key.

Like EncryptSomething this is not a secure process as it uses a public key. It is provided for convenience. See its counterpart, SignSomething.

a. Actors

ServiceProvider

b. Description

c. Notes

WebHome > ProductViewpoint > ProductUseCases > RivetzUseCases > CreateKey

37. Create Key

Create a key pair in the DeviceRivet for either signing and encrypting.

a. Actors

ServiceProvider

b. Description

The primary purpose of Rivetz is to secure and apply keys within endpoint devices. Encryption (privacy) keys or signing (identity) keys are generated using the cryptographic tools in the TEE and securely stored on the device using the TEE's storage key. Bitcoin address keys similarly maintained but have nuances, see CreateBitcoinAccount.

All keys are created in the context of a ServiceProvider. In other words, every key is stored along with the ServiceProviderID that requested its creation. Every key is given a name that is unique with the context of the ServiceProviderID.

When a key is created the rules for its usage are specified in any combination. These are:

- require signed request to apply the key by the key's creator (the ServiceProvider)
- require user confirmation to apply the key through trusted user interface
- require result displayed in TUI

See DecryptSomething and VerifySomething for more discussion at what it means to have the result displayed in TUI.

c. Notes

38. Create Bitcoin Account

Generate a new Wallet Account id in the device hardware

a. Actors

ServiceProvider

b. Description

Like all Riveted keys, the new Bitcoin Account is created within the context of a ServiceProvider and given a name. The ServiceProvider app may hide this name or present it as a feature to the end user.

When creating a Bitcoin Address the ServiceProvider must specify whether the account requires TUI confirmation to sign a transaction.

c. Notes

39. Encrypt Something

Rivetz provides the mechanics for encrypting text or images but expects partners to project the interface for their service, whether it be a messaging application or some other.

Decryption keys can be marked so as to require TUI display of the decrypted object.

MJS> Note that this is distinct from requiring TUI confirmation.

a. Actors

ServiceUser, ServiceProvider

b. Description

The RivetAdaptor will have to have the public key of the target device this is either provided directly by the ServiceProvider or previously recorded in the DeviceRivet during a pairing of devices. On the encryption side, the DeviceRivet need not be involved, as the operation is a public key operation only. Regardless, on the encryption side, the inputs to the function at the HostAdaptor interface (or RivetzEncoder) include:
* Target device ID or target device static public encryption key (the encryption key must be known by the entity performing the encryption) * (Optional) Data to be encrypted

In the simplest instantiation, Rivetz only provides the ECDH operation. When this is done, the data to be encrypted or decrypted is not passed to the Rivetz software, but instead the Rivetz software will simply output the shared secret from the ECDS operation. Then it is up to the external software to perform data encryption using that shared secret.

c. Notes

40. Send a Secure Confirmation Request

Package a short message that will be delivered to the target endpoint device and displayed to the user with secure display if available. The communicated is signed both ways to ensure the confirmation is valid. The message may be an image or text.

a. Actors

ServiceProvider, ServiceUser

b. Description

The value of a secure confirmation request is knowing that there is very little chance (if any) that the message could be confirmed by some other device than the one intended. Further, that the device is displaying a confirmation that could only come the source indicated. To accomplish this requires a registration of keys from both the device and the service provider and a TEE at the device to ensure that nothing untoward is going on when the message is being processed and presented for display in the wild fringe of the network (user's devices).

The service provider will expect to simply declare a message and a target device and await for a response. The keying infrastructure should be independent of all parties and public so as to ensure that only the math is at play as long as the source code is trusted.

c. Notes

41. Sign Something

Given a named key and object reference, return a signed hash of the object

a. Actors

ServiceProvider

b. Description

Note that identity keys will follow the key usage rules as described in CreateKey.

c. Notes

42. Register Device with Rivetz

Before a Rivet can do anything it needs to register with RivetzNet. Registration results in the generation of a unique identity key.

Registration relies on an endorsement from the TrustedApplicationManager to ensure the DeviceRivet is properly executing in a secure environment. (Ideally a key established by the TrustedApplicationManager will locally sign the Device registration key)

a. Actors

TrustedApplicationManager

b. Description

See DeviceEnrollmentProtocol

Registration takes place the first time the RivetAdaptor is invoked and results in a key pair created in the Rivet and the public key shared with RivetzNet. Once a device is registered it will attempt to connect to RivetzNet through a RabbitMQ socket whenever it is live.

1. Device creates local public/private keys

These keys should be locally stored as an identity key to the service provider "Rivetz".

2. Device makes HTTP REST call to rivetz.net requesting registration with signature of public key as unique identifier

RivetzNet needs to test the validity of the request through a protocol provided by the TrustedApplicationManager (TBD).

3. Device receives response showing that it is now registered (or showing that it has previously registered) with its unique device ID and a RabbitMQ queue name to listen for incoming commands

4. Device starts up RabbitMQ to listen to incoming commands on queue specified

c. Notes

43. Sign Bitcoin Transaction

Given a fully formed bitcoin transaction (where the origin account is owned by the target device hardware), sign the transaction and return it. In most cases this should also involve prompting the user for confirmation with secure display, if available, or at least common display otherwise.

a. Actors

ServiceProvider, ServiceUser

b. Description

c. Notes

44. Create Local User

Establish a local entity that can authorize use of the Rivet in cases where no ServiceProvider authorization is given

a. Actors

Select/create Actors from ProductActors

* DeviceRivet

* TEEAdapter

* Rivetz.net (Optional)

b. Description

In order to enable fast and easy use of the DeviceRivet, the DeviceRivet may allow the creation of a "local user". The LocalUser is defined to be an entity that is not an authorized ServiceProvider, but that is allowed to access the DeviceRivet in some capacity. While a ServiceProvider may be allowed to create and manage bitcoin keys and provide other services, the LocalUser may only be authorized to perform certain operations. These operations may include:

* Creating and using encryption keys

* Creating and using signature keys

The properties of a local user are as follows:

- The authorization for a LocalUser will initially be held on the local platform, but could later be protected elsewhere

- The LocalUser is optionally authorized by Rivetz.net

- The LocalUser may be hidden from the actual human user or application. It may be managed within the RivetAdaptor

- Protection of the authorization for the LocalUser can be enhanced over time to include encryption with a user password or use of some other protection mechanism

- From an application perspective, the HostAdaptor provides an interface that makes the notion of the LocalUser transparent, other than the fact that the keys associated with the LocalUser are not accessible through any interface other than through the HostAdaptor

We should be careful in considering the name of the "local user", as this is a user from the DeviceRivet perspective, but not necessarily from the external perspective. One concept is that the local user is handled by the TEEAdapter. The TEEAdapter establishes a shared secret with the DeviceRivet or creates a public key that authorizes the local user with the DeviceRivet.

c. Notes

45. Local User

This is an entity that can access the DeviceRivet without participation from a formal ServiceProvider. That is, this is a role that is different from the typical service provider and it may be expected that there could be a different LocalUser for each DeviceRivet, which is only able to access one particular DeviceRivet.

Some decisions should be made about the provisioning of a LocalUser, but one possibility is that Rivetz.net authorizes the LocalUser during a provisioning step in the same manner that might be done with a typical ServiceProvider (e.g. through a "pairing" operation). If this is the case, Rivetz can still maintain control over who can access DeviceRivet services and also, down the road, provide some strong protections over access to the LocalUser role (by ensuring the authorization for the LocalUser is strongly protected and controlled by some trusted entity).

A decision should also be made about the manner in which the LocalUser is authorized. For simplicity, we could require that operations by the LocalUser require the same kind of authorization as operations from a ServiceProvider (e.g. through a signature operation) or, in the short term, we could simply allow the LocalUser to utilize a share secret (e.g. a password, passphrase or random value).

- Local User

46. Register Device with Service Provider

A service provider needs to have their ServiceProviderID and public identity key registered with a device before that device will respond to any requests.

Even in cases where the named key (identity, privacy or coin) does not require a signed request, the ID of the requesting party must be known to the the device. RivetzNet is responsible for endorsing the relationship between a device and a service provider. In this way we maintain some control over the ecosystem. It also enables us to provide services to end users regarding the use, backup and migration of service provider keys.

a. Actors

ServiceProvider

b. Description

1. Local service provider app makes request to RivetAdaptor for device pointer
2. Device makes HTTP REST call to RivetzNet with new device pointer and device ID (NOTE: need authentication here....could use public key or API key, similar to above) as well as public key
3. Response from server includes RabbitMQ queue to await incoming service provider's public key
4. Service provider passes device pointer to their servers
5. Service provider makes HTTP REST call with device pointer and SP's public key
6. Response to service provider includes device public key
7. Service provider's public key is pushed to device

c. Notes

47. Decrypt Something

Given an encrypted object and a key name, decrypt the object either for TUI display or to return to the requester.

a. Actors

ServiceProvider

b. Description

When a privacy key pair is created it needs to be marked with key usage rules that specify whether the request needs to be signed and/or confirmed by the user through the TUI. Further, the key can be designated as for TUI Display only meaning that anything it decrypts stays in secure world.

C. Notes

CLAIMS

What is claimed is:

1. A computer-implemented method of verifying device integrity of a user device in a block chain communication network comprising:

    in preparation for delivering an electronic transaction in the block chain network, implementing a device integrity verification process as part of the transaction including:

        performing an internal validation of the integrity of the device execution environment from a root of trust in the user device; and

        requiring an electronic signature, such that a verification of the integrity of the signature is applied to the block chain transaction;

        wherein verification of the integrity of the signature is based on a determination of whether the execution environment of the device is in a known good condition including:

            based on the integrity of the signature, allowing the transaction to proceed or requesting a remediation authority to verify that the electronic transaction as intended by the user is allowed to proceed even if it is determined that the execution environment of the device is not in a known good condition.

2. The method of Claim 1 wherein verification of the integrity of the signature includes:

    transmitting a root of trust instruction to the block chain network for processing, such that at least a portion of the block chain network responds by requiring multiple electronic signatures in order to accept the electronic transaction including:

        creating within the execution environment of the device, an instruction from a root of trust in the user device;

        requiring a first electronic signature that corresponds to the root of trust instruction, such that a verification of the integrity of the signature is applied to the block chain transaction; and responding to the first electronic signature by verifying the integrity of the signature based on a determination of whether the execution environment of the device is in a known good condition including:

comparing the signature with a previously recorded reference value;

if the signature matches the previously recorded reference value, then allowing the transaction to proceed; and

if the signature does not match the previously recorded reference value, requesting a third party out of band process to verify that the electronic transaction as intended by the user is allowed to proceed even if it is determined that the execution environment of the device is not in a known good condition.

3. The method of Claim 1 wherein verifying the integrity of the signature includes:

the device providing the electronic signature based on a determination of whether the execution environment of the device is in a known good condition;

allowing the transaction to proceed if the device provides the electronic signature;

allowing the transaction as intended by the user to proceed even if it is determined that the execution environment of the device is not in a known good condition if the remediation authority provides the signature.

4. The method as in Claim 2 wherein the out of band process further includes using an N or M cryptographic key function to confirm that at least one of: an intent of the user meets predetermined requirements, or the device integrity meets predetermined requirements, or an additional process meets predetermined requirements.

5. The method as in Claim 2 wherein the reference value is generated during a registration process performed by the owner of the device platform.

6. The method as in Claim 2 wherein the reference value is generated based on a birth certificate assigned to the device, wherein the birth certificate is generated by the manufacturer or creator of the device, the manufacturer or creator of the execution environment of the device and/or the manufacturer or creator of an application on the device.

7. The method as in Claim 2 wherein the reference value includes a signature of at least one of the manufacturer or creator of the device, the manufacturer or creator of the execution environment of the device and/or the manufacturer or creator of an application on the device.

8. The method as in Claim 2 wherein the third party out of band process returns a token in response to the request to verify the transaction.

9. The method as in Claim 2 further allowing the electronic transaction to be completed within a certain period of time if the signature does not match the previously recorded reference value.

10. The method as in Claim 2 wherein verifying that the intended electronic transaction is allowed to proceed even if it is determined that the execution environment of the device is not in a known good condition is based on a period of time between the registration of the reference value and the transaction and/or the amount of the transaction.

11. The method as in Claim 10 wherein transactions above a threshold amount are allowed to proceed if the period of time meets predetermined requirements.

12. The method as in Claim 11 wherein allowing the transaction above a certain amount is based on a minimum number of previously allowed transactions.

13. The method as in Claim 1 further comprising using a display device indicating to the user whether device integrity meets minimum predetermined requirements and further actions to be taken.

14. The method as in Claim 1 further including notification to a third party of the transaction, wherein in response to the notification, the third party records the transaction and a state of the device.

15. The method as in Claim 14 wherein the third party records measurements associated with the device integrity for future analysis of the transaction.

16. The method as in Claim 14 further assuring the privacy of the record including cryptographically obfuscating the record such that the record is made available only to authorized third parties.

17. A computer-implemented system of verifying device integrity of a user device in a block chain communication network comprising:

a block chain communication network;

a user device in the block chain network;

an electronic transaction in the block chain network;

a device verification process implemented as a part of the transaction in preparation for delivery of the electronic transaction in a block chain network, the implementation further comprising:

an internal validation of the integrity of the device execution environment performed from a root of trust in the device;

an electronic signature, such that a verification of the integrity of the signature is applied to the block chain transaction;

wherein verification of the integrity of the signature is based on a determination of whether the execution environment of the device is in a known good condition including:

based on the integrity of the signature, allowing the transaction to proceed or requesting a remediation authority to verify that the electronic transaction as intended by the user is allowed to proceed even if it is determined that the execution environment of the device is not in a known good condition.

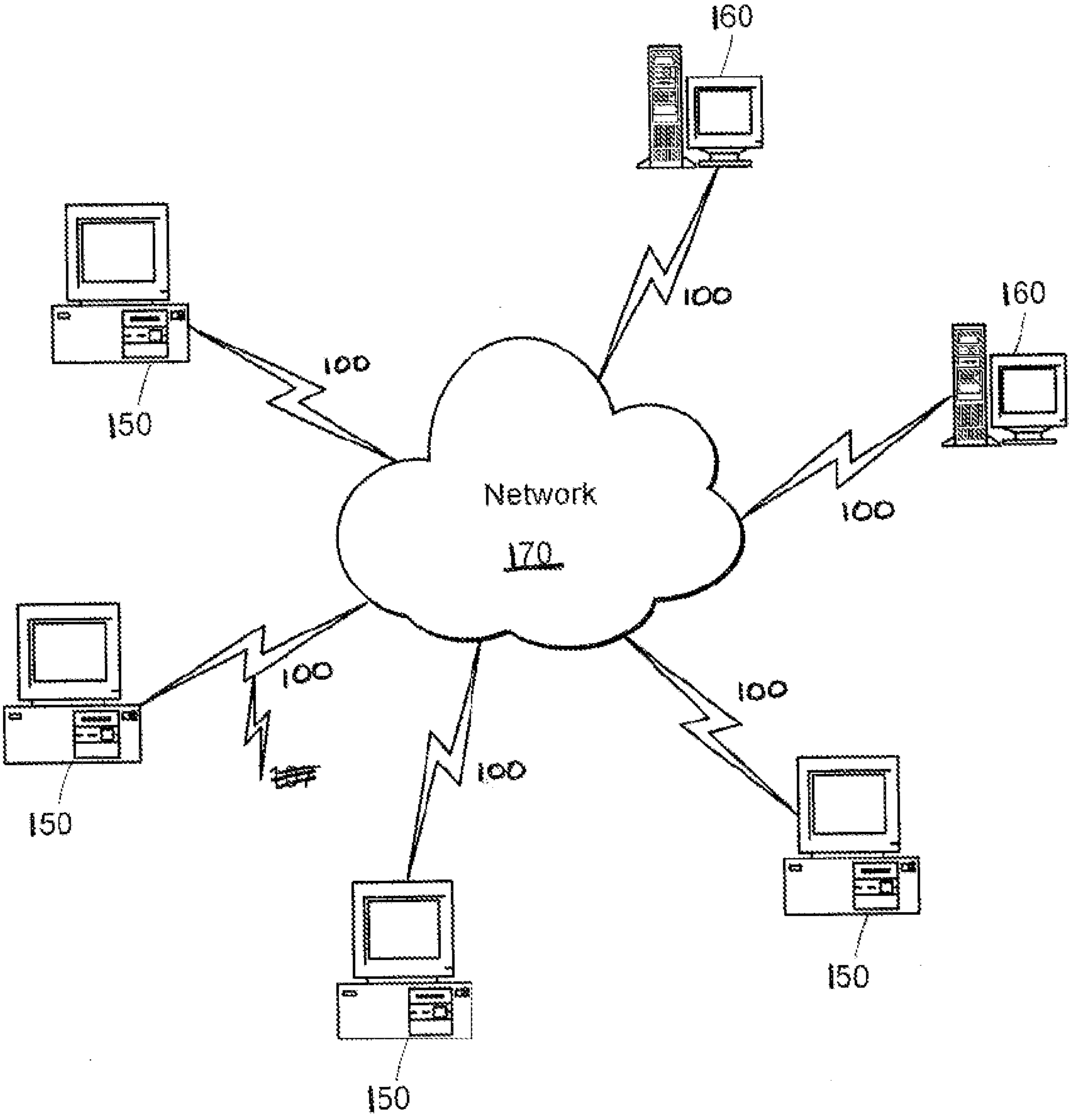


FIG. 1A

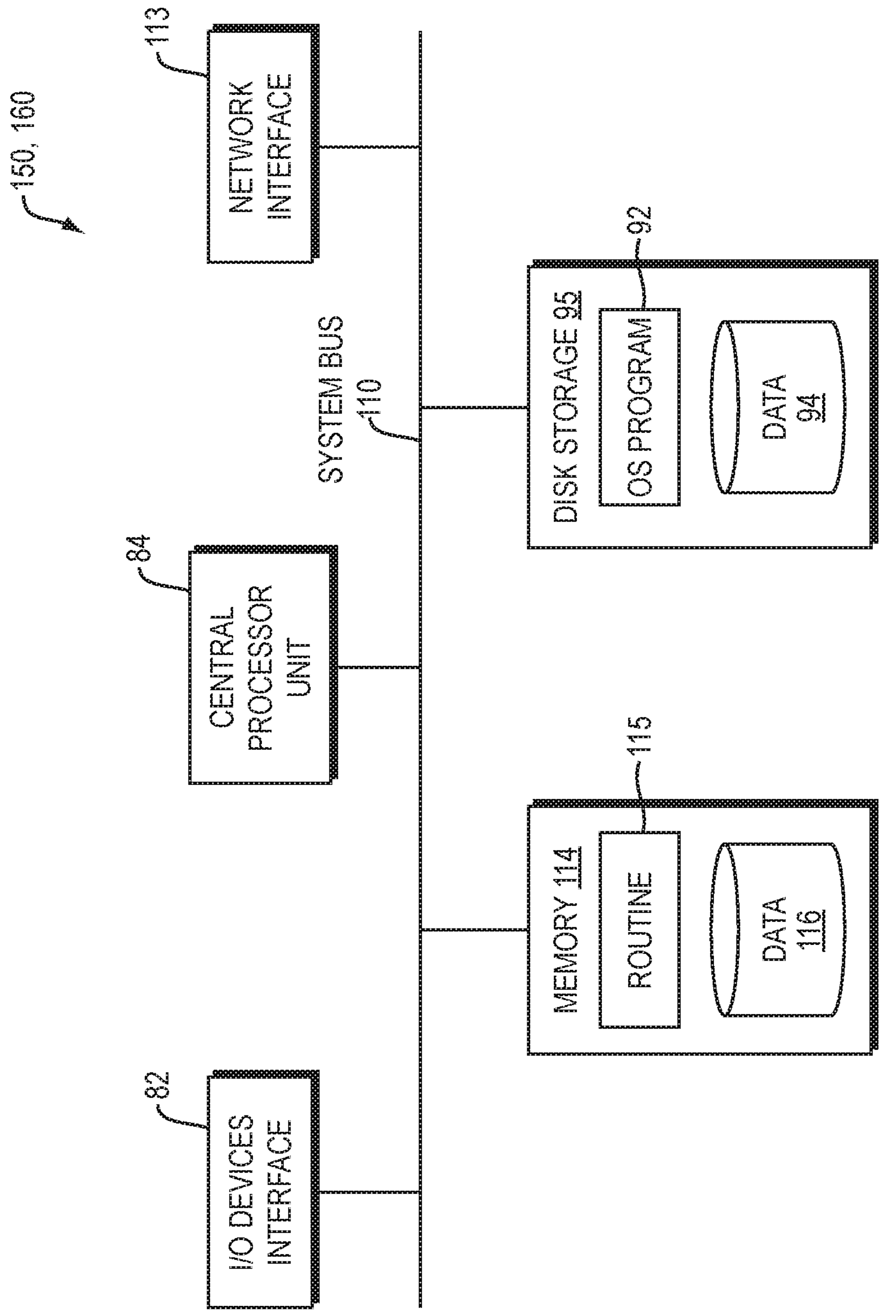

150, 160
NETWORK INTERFACE
113
CENTRAL PROCESSOR UNIT
84
I/O DEVICES INTERFACE
82
SYSTEM BUS
110
DISK STORAGE 95
92
OS PROGRAM
DATA 94
MEMORY 114
115
ROUTINE
DATA 116

FIG. 1B

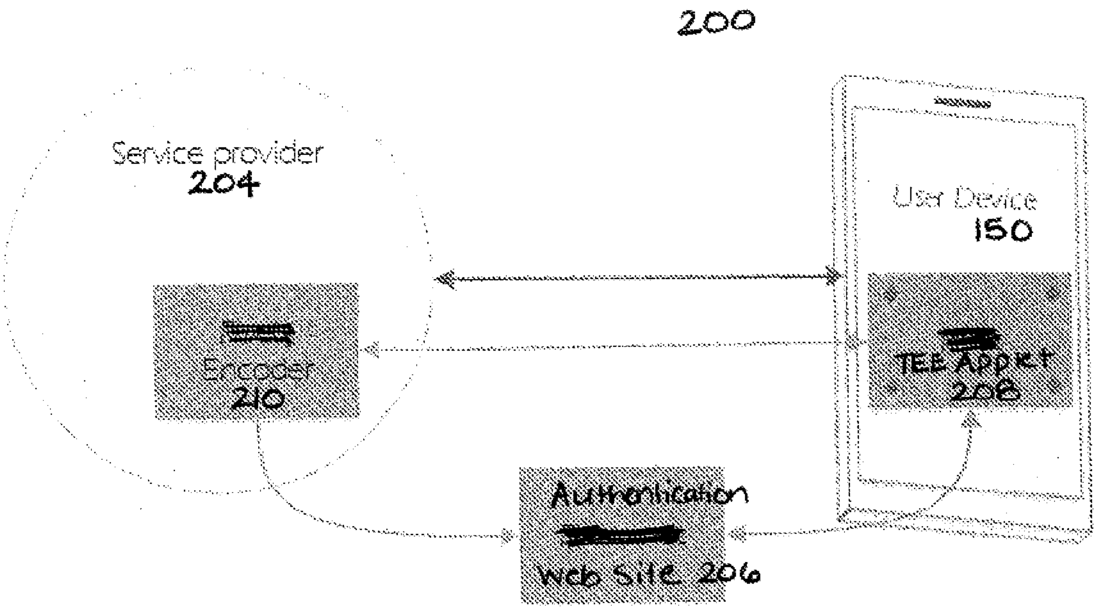


FIG. 2A: example device authentication system according to the invention.

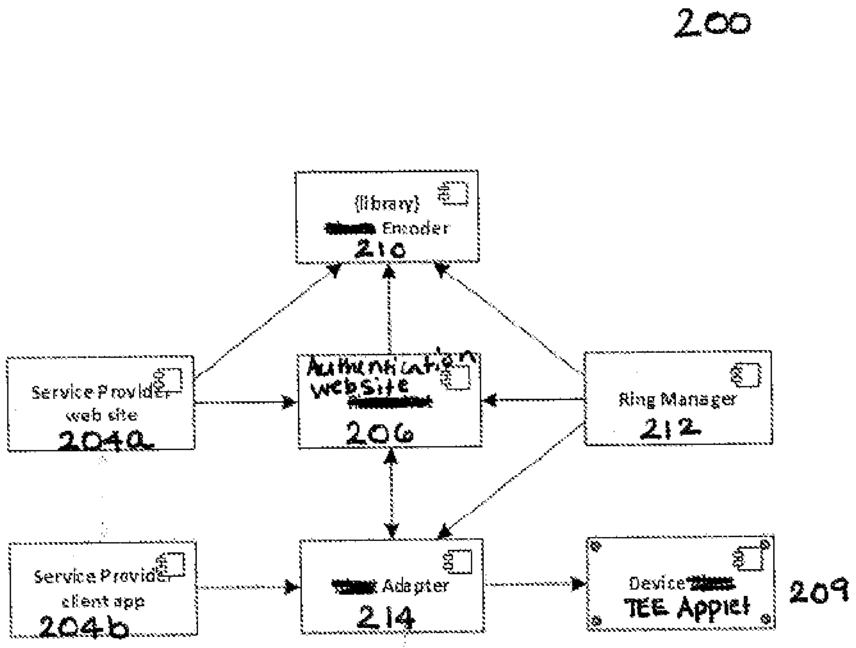


FIG. 2B: example device authentication system according to the invention.

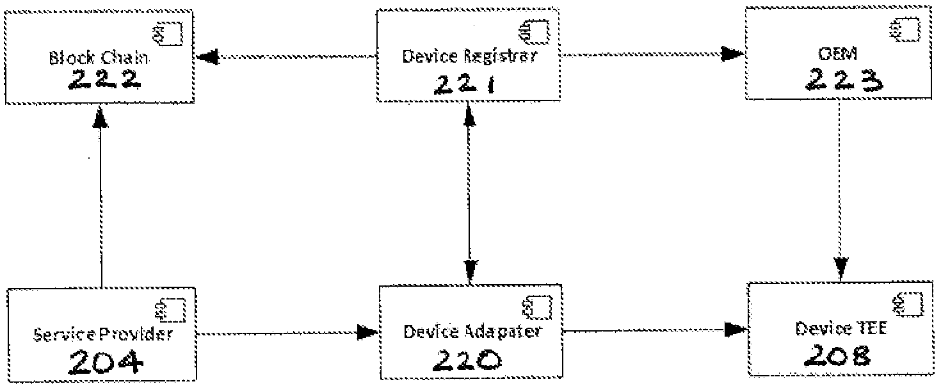


FIG. 2C: Components of an embodiment of the invention

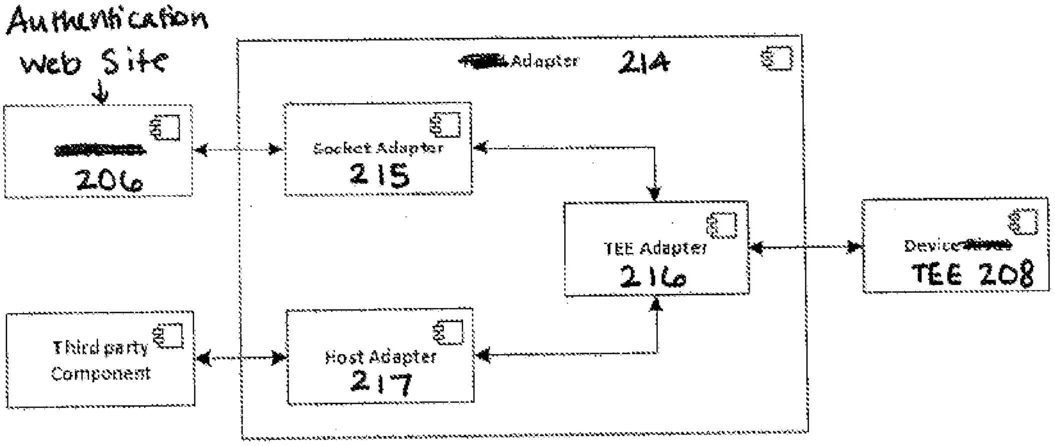


FIG. 2D: Authentication System Adaptor and its outward and inward looking interfaces.

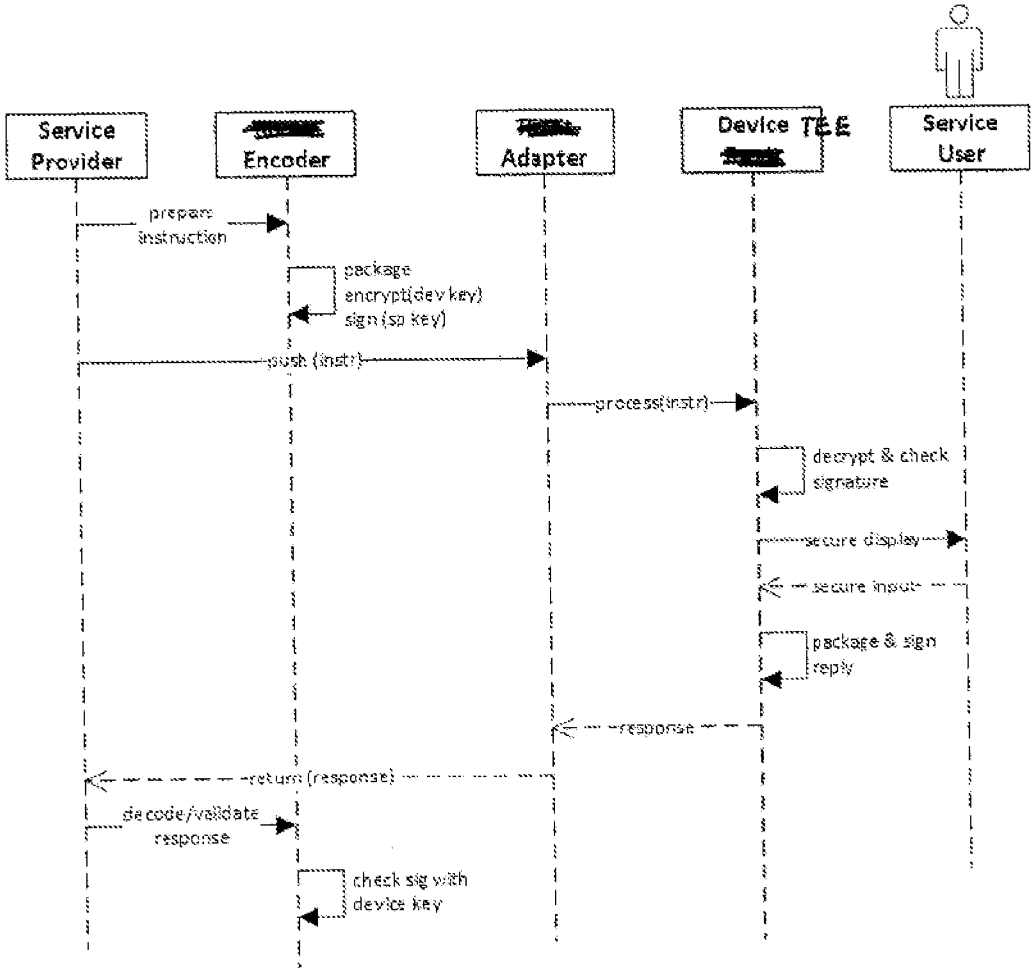


FIG. 3A: sequence of packaging and delivering an instruction by the Encoder.

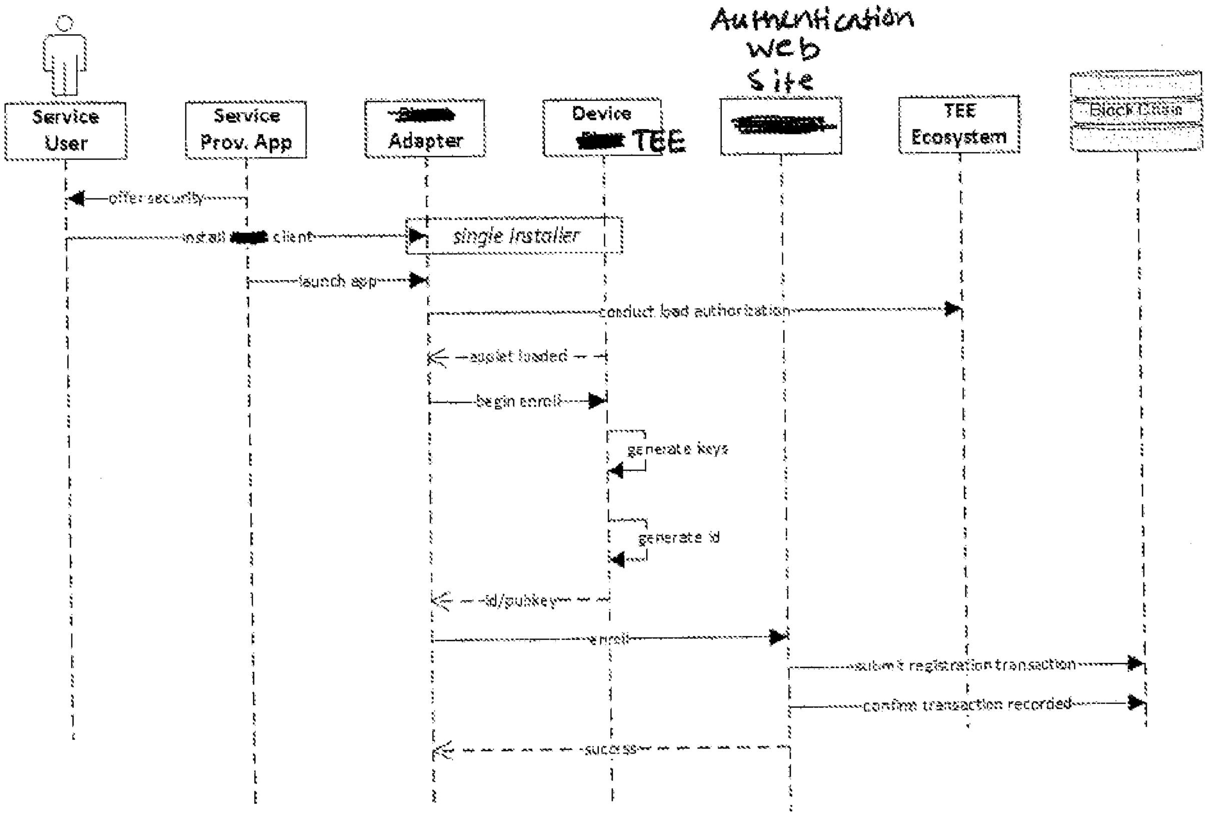


FIG 3B: Device enrollment process according to an embodiment of the invention.

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US 16/23142

A. CLASSIFICATION OF SUBJECT MATTER
IPC(8) - G06F 11/30, G06F 12/14 (2016.01)
CPC - G06F 21/72
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC(8): G06F 11/30, G06F 12/14 (2016.01)
CPC: G06F 21/72

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
USPC: 713/189, 713/190, 713/176 (Keyword limited; terms below); IPC(8): G06F 11/30; G06F 12/14 (2016.01) (Keyword limited; terms below); CPC: G06F21/72, G06F21/10, G06F2221/2107, H04L9/08, H04L63/0428 (Keyword limited; terms below)

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
PatBase; Google (Scholar, Patents, Web)
Terms used: blockchain transaction device verification integrity authentication root trust internal signature condition root "birth certificate"

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X --- Y	US 2014/0357295 A1 (SKOMRA et al.), 04 December 2014 (04.12.2014), entire document, especially Abstract; para [0013], [0025], [0065], [0071], [0095], [0135], [0139], [0153], [0155], [0160], [0173], [0182], [0212], [0234], [0237], [0262]-[0263], [0269]	1-7, 13-17 -------------- 8
Y	US 2015/0081566 A1 (SLEPININ), 19 March 2015 (19.03.2015), entire document, especially Abstract; para [0057]	8
A	US 2006/0129825 A1 (SALOMON et al.), 15 June 2006 (15.06.2006), entire document	1-17
A	US 2011/0307703 A1 (OGG et al.), 15 December 2011 (15.12.2011), entire document	1-17
A	US 2014/0136838 A1 (MOSSBARGER), 15 May 2014 (15.05.2014), entire document	1-17
A	US 2014/0279526 A1 (JACKSON), 18 September 2014 (18.09.2014), entire document	1-17

□ Further documents are listed in the continuation of Box C. □

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search 06 June 2016 (06.06.2016)	Date of mailing of the international search report 26 JUL 2016
Name and mailing address of the ISA/US Mail Stop PCT, Attn: ISA/US, Commissioner for Patents P.O. Box 1450, Alexandria, Virginia 22313-1450 Facsimile No. 571-273-8300	Authorized officer: Lee W. Young PCT Helpdesk: 571-272-4300 PCT OSP: 571-272-7774

摘要

公开了在接受区块链交易之前提供对未知客户端设备的全面验证的系统和方法，其将对区块链交易提供进一步的安全性。设备的健康可以在进行电子交易之前被认证。在一些实施方式中，全面的设备完整性验证的自动化被提供为区块链交易的一部分。本发明的某些方面使得能够信任设备。一些实施方式在与设备的可靠关系可以有利于与终端用户的更安全、更容易且更强大的关系的基本前提下操作。实现上述需要明确得知参与当前交易的设备是先前交易中的相同设备。